Freescale Semiconductor, Inc.

ICD-12 Target Interface

Revised 12/12/2002

Metrowerks, the Metrowerks logo, and CodeWarrior are registered trademarks of Metrowerks Corp. in the US and/or other countries. All other tradenames and trademarks are the property of their respective owners.

Copyright © Metrowerks Corporation. 2003. ALL RIGHTS RESERVED.

The reproduction and use of this document and related materials are governed by a license agreement media, it may be printed for non-commercial personal use only, in accordance with the license agreement related to the product associated with the documentation. Consult that license agreement before use or reproduction of any portion of this document. If you do not have a copy of the license agreement, contact your Metrowerks representative or call 800-377-5416 (if outside the US call +1-512-996-5300). Subject to the foregoing non-commercial personal use, no portion of this documentation may be reproduced or transmitted in any form or by any means, electronic or mechanical, without prior written permission from Metrowerks.

Metrowerks reserves the right to make changes to any product described or referred to in this document without further notice. Metrowerks makes no warranty, representation or guarantee regarding the merchantability or fitness of its products for any particular purpose, nor does Metrowerks assume any liability arising out of the application or use of any product described herein and specifically disclaims any and all liability. **Metrowerks software is not authorized for and has not been designed, tested, manufactured, or intended for use in developing applications where the failure, malfunction, or any inaccuracy of the application carries a risk of death, serious bodily injury, or damage to tangible property, including, but not limited to, use in factory control systems, medical devices or facilities, nuclear facilities, aircraft navigation or communication, emergency systems, or other applications with a similar degree of potential hazard**.

USE OF ALL SOFTWARE, DOCUMENTATION AND RELATED MATERIALS ARE SUBJECT TO THE METROWERKS END USER LICENSE AGREEMENT FOR SUCH PRODUCT.

How to Contact Metrowerks

Corporate Headquarters	Metrowerks Corporation 7700 West Parmer Lane Austin, TX 78729 U.S.A.
World Wide Web	`http://www.metrowerks.com`
Sales	Voice: 800-377-5416 Fax: 512-996-4910 Email: sales@metrowerks.com
Technical Support	Voice: 800-377-5416 Email: support@metrowerks.com

Table of Contents

1 Overview 7

1.1 About this guide . . . 7
1.2 Highlights . . . 8
1.3 Requirements . . . 8

2 Getting Started with CodeWarrior and ICD-12, and more... 9

2.1 Technical Considerations about ICD-12 Target Interface . . . 9
2.2 First Steps with CodeWarrior and the ICD-12 Target Interface . . . 9
2.3 How to load an Application in Flash EEPROM . . . 13
2.4 How to Set Breakpoints in Non Volatile Memory . . . 20

3 ICD-12 Target Interface Manual 23

3.1 Introduction. . . . 23
3.2 Interfacing Your System with the Target . . . 24
3.2.1 Hardware Connection . . . 24
3.2.2 Loading the ICD-12 Target Interface . . . 25
3.3 ICD-12 Target Interface Menu Entries . . . 27
3.3.1 Load... . . . 28
3.3.2 Reset . . . 28
3.3.3 Communication... or Connect... . . . 28
3.3.4 Command Files . . . 28
3.3.5 Set MCU Type... . . . 28
3.3.6 Set BDM Speed... . . . 28
3.3.7 Set Hardware BP... . . . 29
3.3.8 Set Bank... . . . 29
3.3.9 Unsecure... . . . 29
3.3.10 Flash... . . . 29
3.4 ICD-12 Target Interface Dialogs. . . . 29
3.4.1 Communication Device Specification . . . 30
3.4.2 MCU Selection . . . 31
3.4.3 Set MCU Speed . . . 32
3.4.4 Non Volatile Memory Control Dialog . . . 33
3.5 Status Bar Information for the ICD-12 Target Interface. . . . 34
3.5.1 Status Messages . . . 34
3.5.2 Stepping and Breakpoint Messages . . . 35

For More Information: www.freescale.com

3.6 ICD-12 Target Interface Default Environment . . . 36
3.6.1 Default Target Setup . . . 36
3.6.2 ICD-12 Target Interface Environment Variables . . . 36
3.7 ICD-12 Target Interface Command Line commands . . . 44

4 HC12 and HCS12 Banked Memory support 51
4.1 Banked Memory Location Dialog Box . . . 51
4.1.1 PPAGE index tab . . . 52
4.1.2 DPAGE index tab . . . 52
4.1.3 EPAGE index tab . . . 53
4.1.4 Various index tab (not all Target Interfaces) . . . 54
4.2 Associated Commands . . . 55
4.3 Associated Environment Variables . . . 58

5 Unsecure Motorola HCS12 derivatives 61
5.1 Unsecure derivative dialog box . . . 61
5.2 Unsecure Command File . . . 63
5.3 Associated Commands . . . 66
5.4 Associated Environment Variables . . . 67

6 Target Interface Commands Files 69
6.1 Target Interface Associated Command Files . . . 69
6.1.1 Startup Command File . . . 70
6.1.2 Reset Command File . . . 70
6.1.3 Preload Command File . . . 70
6.1.4 Postload Command File . . . 71
6.1.5 Vppon Command File . . . 71
6.1.6 Vppoff Command File . . . 71
6.2 Command Files dialog . . . 72
6.3 Associated Commands . . . 73
6.4 Associated Environment Variables . . . 75

7 On-Chip Hardware Breakpoint Module 77
7.1 Hardware Breakpoint Configuration dialog . . . 77
7.1.1 Disabled mode . . . 79
7.1.2 Automatic (controlled by debugger) mode . . . 79
7.1.3 User Controlled mode . . . 81

7.2 Associated Commands 84
7.3 Associated Environment Variables 87

8 Running Standard EVBs with a BDM pod 91
8.1 Connection between a BDM pod and an EVB board 91
8.2 MC68HC812A4EVB Evaluation Board 92
8.3 MC68HC912B32EVB Evaluation Board 93
8.4 MC68HC912D60EVB Evaluation Board 93
8.5 MC68HC912DG128EVB Evaluation Board 93
8.6 EVB912DP256 Evaluation Board 94
8.7 Axiom CML12S-DP256 94

Index 95

For More Information: www.freescale.com

1

Overview

1.1 About this guide

This document includes information to become familiar with the *ICD-12* Target Interface and to help you understand how to use this Target Interface. This document is divided into following sections:

- The Introduction section introduces the *ICD-12* Target Interface concept.
- The Getting Started with CodeWarrior and ICD-12, and more... section gives answers for common questions and describes how to use advanced features of the *ICD-12* Target Interface. You will find out First Steps with CodeWarrior and the ICD-12 Target Interface, How to load an Application in Flash EEPROM and How to Set Breakpoints in Non Volatile Memory.
- The Interfacing Your System with the Target section contains information about the connection between the *P&E Cable12* and *BDM-MULTILINK* and the debugger.
- The ICD-12 Target Interface Menu Entries section gives a description of the *ICD-12* Target Interface specific menu entries.
- The ICD-12 Target Interface Dialogs section gives a description of the *ICD-12* Target Interface specific dialog boxes.
- The Status Bar Information for the ICD-12 Target Interface section describes the status bar messages for the *ICD-12* Target Interface.
- The ICD-12 Target Interface Default Environment section lists all the variables used by this Target Interface to store the configuration.
- The ICD-12 Target Interface Command Line commands section lists all the commands specific to this Target Interface.
- The Unsecure Motorola HCS12 derivatives section describes how to use this Target Interface to unsecure a *HCS12* derivative.
- The Running Standard EVBs with a BDM pod section describes how to connect to standard evaluation boards using the *ICD-12*.

- The On-Chip Hardware Breakpoint Module section explains how to use the Hardware Breakpoint Configuration dialog to set hardware breakpoints and watchpoints.
- The Index contains all keywords of the *ICD-12* Target Interface.

1.2 Highlights

- The *ICD-12* Target Interface allows you to debug with the *P&E Cable12* and *P&E BDM-MULTILINK*.
- Debugging in Flash is supported (flash programming and hardware breakpoints).
- Periodical update for variables and memory is available for the *ICD-12* Target Interface.
- The *ICD-12* Target interface offers the possibility to unsecure a secured *Motorola HCS12* derivative (see Unsecure derivative dialog box).
- Breakpoint insertion while the target is running (intrusive).
- At connection, the Target Interface can detect if the target processor is running. The target processor is not reset and can be halted later (see Communication Device Specification dialog).
- Register block moving handling: on *HCS12* derivatives, the Target Interface can automatically track device Registers block ($0-$3FF) relocated via INITRG register setup. For *HC12* derivatives, a new command has been implemented to specify the new location of the register block: REGBLKADR.

1.3 Requirements

In order to use the *P&E Cable12* or *P&E BDM-MULTILINK*, the drivers from *P&E* must be installed on the host computer. A copy of those drivers are installed while installing *Metrowerks CodeWarrior for HC12 Studio*.

For more information, please see section Interfacing Your System with the Target.

2

Getting Started with CodeWarrior and ICD-12, and more...

Thanks for choosing *CodeWarrior*. This section guides you through installation, licensing/registration and first steps with CodeWarrior and the *ICD-12* Target Interface. It does not replace all the documentation provided, but gives you a good starting point.

2.1 Technical Considerations about ICD-12 Target Interface

A parallel cable should be used for the communication between the *P&E Cable12* or *BDM-MULTILINK* and the host computer.

The communication protocol between the *P&E Cable12* or *BDM-MULTILINK* and host is fully handled by the *unit_12z.dll* Target driver which is automatically loaded with the *ICD-12* Target Interface.

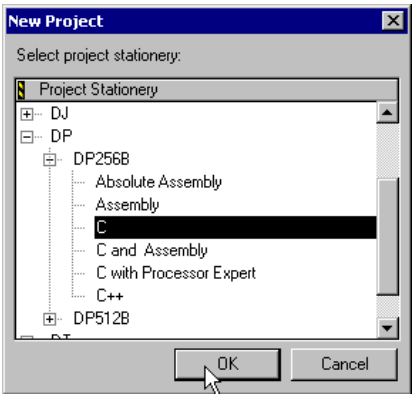
2.2 First Steps with CodeWarrior and the ICD-12 Target Interface

1.Run the *CodeWarrior IDE* with the shortcut created in the program group.

2.Choose the menu **File > New** to create a new project from a stationery.

3.Select *HC12 Stationery*, type in a project name and specify the project location. Press *OK*.

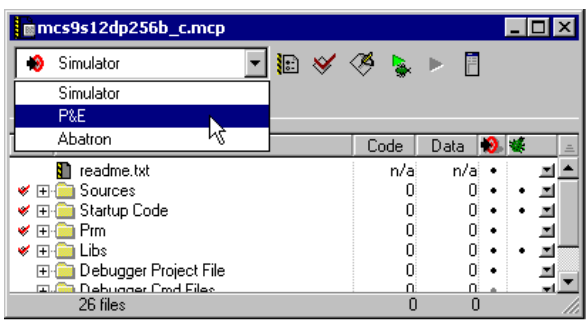
Figure 2.1 New Project dialog



4.From the available project stationery, select the stationery which corresponds to the derivative connected to your hardware and press *OK*.

5.Now you have a project with all files set up. Use the menu **Project > Set Default Target > P&E** to use the *ICD-12* Target Interface.

Figure 2.2 Selecting P&E Build Target



6.Make sure that the board hardware settings are the default settings for the BDM mode (please see your user manual).

7.Then connect the *P&E Cable12* or *BDM-MULTILINK* to the parallel communication port `LPT1` or `LPT2` of your PC, using a parallel cable.

8.Power supply the board with the appropriate voltage.

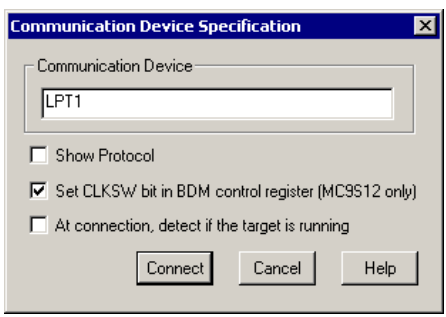
9.Start the debugger using the menu **Project > Debug**, which loads automatically the application into the target. The first time, or if the code has been changed, the application is rebuild.

10.If the "*Error*" message shown in Figure 2.3 is displayed, it means that the debugger did not found the board on your `LPT1` port.

Figure 2.3 The board could not be found on your LPT1 port.

11.Click *Ok*. You will be prompted to specify the communication port in the dialog shown in Figure 2.4.

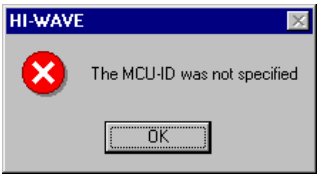
Figure 2.4 Communication Device Edit Box.



12.In the *Communication Device* edit box, type the correct parallel port. Click the *Connect* button. The debugger tries then to connect to the target.

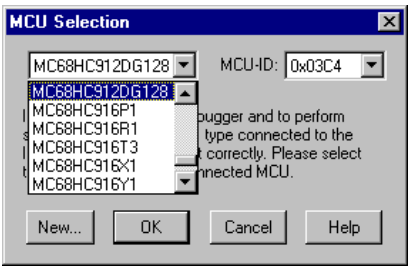
13.If the Target Interface is started for the first time in the directory, the dialog shown in Figure 2.5 is displayed:

Figure 2.5 MCU-ID not specified.



14.Close it with *OK*. The *MCU selection* dialog is then displayed (see Figure 2.6). In the drop down list box, select the derivative which is on your target system.

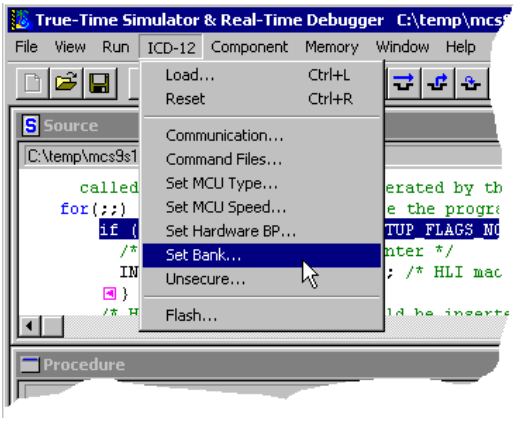
Figure 2.6 MCU Selection.



15.Close the *MCU Selection* dialog clicking *OK*. Connection to target is established.

16.The debugger's main menu contains a *ICD-12* entry (see Figure 2.7) from where your can reset the application when selecting *Reset*, set the communication parameters when selecting *Communication...* or load a `.ABS` application to debug when selecting *Load...*, etc..

Figure 2.7 ICD-12 Menu Entry.



2.3 How to load an Application in Flash EEPROM

The *ICD-12* Target Interface offers the possibility to directly program the on-board *Flash EEPROM* of your MCU.

The following steps show you how to load an application in the on-chip Flash EEPROM of a derivative debugged using the *ICD-12* Target Interface.

Those steps imply the target board is correctly power supplied and the programming voltage for flash programming is applied to the target system.

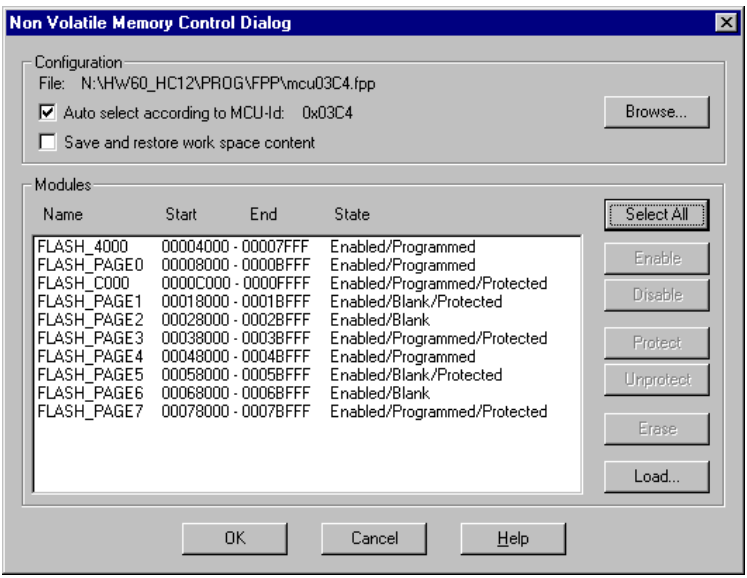
CAUTION When using stationery, the application is automatically loaded in flash using the Preload Command File and Postload Command File. When *Flashing using the Non Volatile Memory Control* dialog,

please make sure the *Preload* and *Postload command files* are disabled.

1.Make sure the *Preload* and *Postload command files* are disabled (in the *ICD-12 Command Files* dialog, menu command **ICD-12 > Command Files ...**).

1.Once the debugger is started and the connection with the board is correctly established, choose **ICD-12 > Flash**. The *Non Volatile Memory Control Dialog* dialog is then opened, as shown in Figure 2.8.

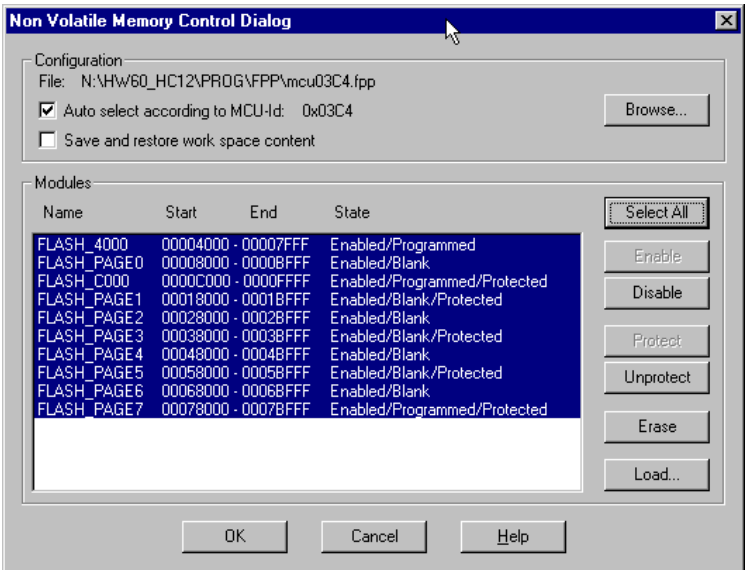
Figure 2.8 Non Volatile Memory Control Dialog.



2.Check that the correct *FPP* file is selected in the *Configuration* panel. This *FPP* file has to match with your derivative MCU-ID, set up through the MCU Selection dialog.

3.Click on the *Select All* button as shown in Figure 2.9. All the non volatile memory modules present in the *Module* list are selected.

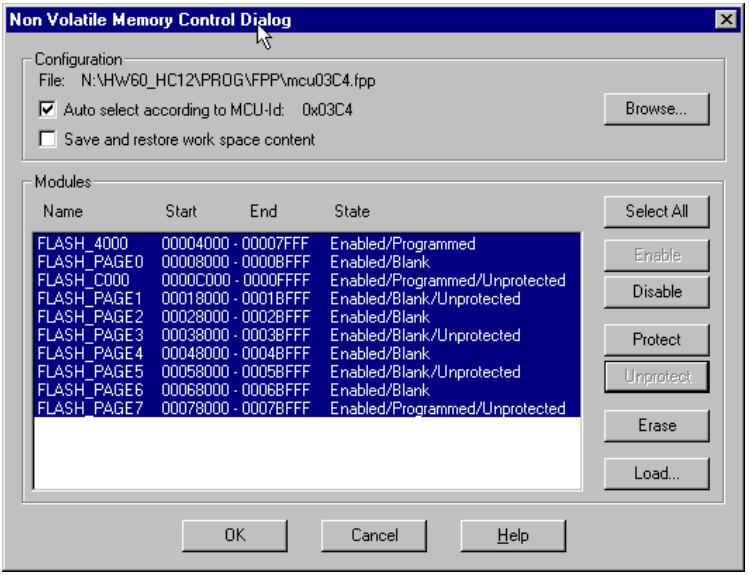
Figure 2.9 Selection Of All The Non Volatile Memory Modules.



4.Click the *Unprotect* button.

The module which were marked *Protected* in the *Modules* list are now marked *Unprotected* (see Figure 2.10).

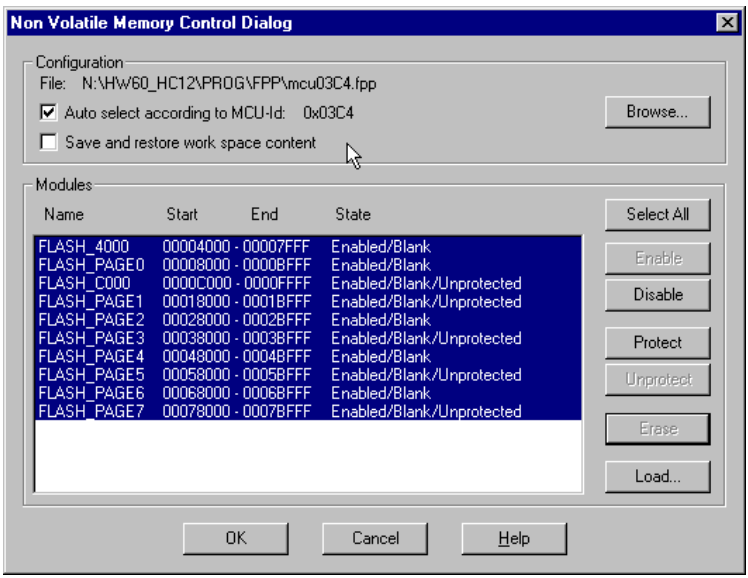
Figure 2.10 Unprotecting The Non Volatile Memory Modules.



5.Click the *Erase* button.

The module which were marked *Programmed* in the *Modules* list are now marked *Blank*, as shown in Figure 2.11.

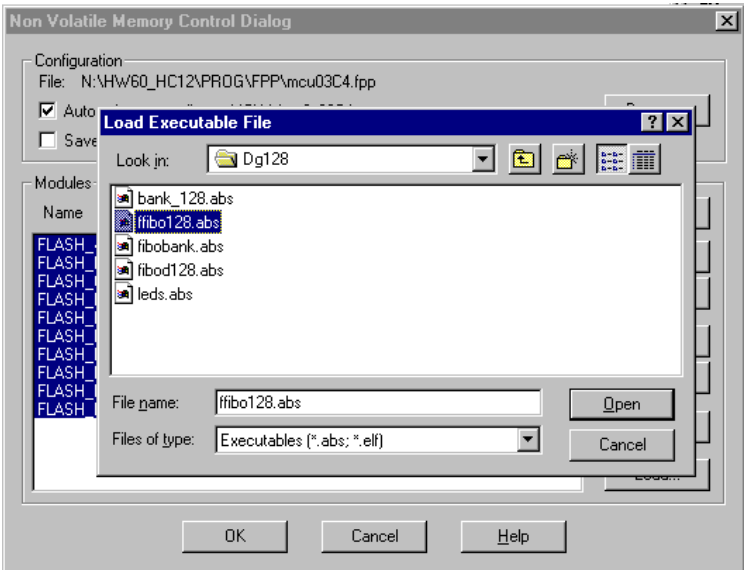
Figure 2.11 Erasing The Non Volatile Memory Modules.



6.Click *Load...*

The *Load Executable File* dialog is opened (see Figure 2.12).

Figure 2.12 Opening The Load Executable File Dialog



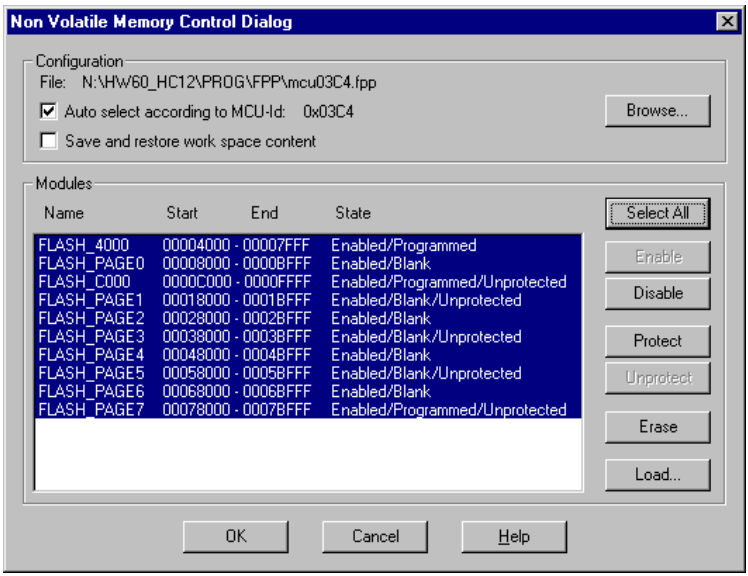
7.Select the file you want to load in Flash memory.

8. Click *Open*.

The *Load Executable File* dialog is closed. Wait a few seconds. Some of the modules which were marked *Blank* in the *Modules* list are now marked *Programmed*, as shown in Figure 2.13.

The application has been written in the Flash EEPROM.

Figure 2.13 Programming The Non Volatile Memory Modules.

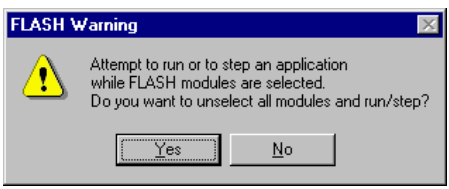


9. Close the *Non Volatile Memory Control Dialog* dialog.

10.Choose **ICD-12 > Reset**.

Figure 2.14 shows the *FLASH Warning* dialog box which is displayed.

Figure 2.14 Flash Warning Dialog



11.Choose **Run > Start/Continue** or click . The application is started.

12.Choose **Run > Halt** or click .The execution of the program is stopped.

2.4 How to Set Breakpoints in Non Volatile Memory

Once an application has been loaded in non volatile memory, the hardware breakpoint module must be configured in order to allow you to debug using breakpoints.

During the first connection, the hardware breakpoints module settings are resolved according to the specified derivative.

Afterwards, if the user change the derivative, it is his responsibility to setup correctly the hardware breakpoints mechanism for the project.

1.Select **ICD-12 > Set Hardware BP...** menu entry. The Hardware Breakpoint Configuration dialog is displayed, as shown in Figure 2.15.

2.Select the *Automatic (controlled by debugger)* mode in the *Mode* list.

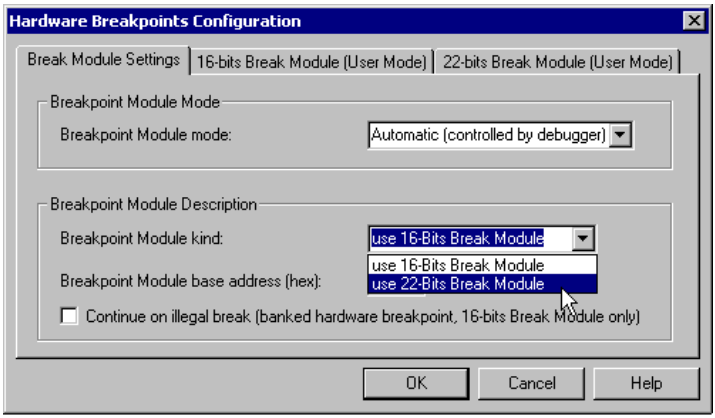
3.Select the *Breakpoint Module kind* of the hardware breakpoint module supported by the derivative currently connected: “use 16-Bits Break Module” for a *Motorola HC12* derivative and “use 22-Bits Break Module” for a *Motorola HCS12* derivative.

4.In order to set the debugger correctly, the address of the hardware breakpoint module must be set in the *Module base address* edit box.

- The *Module base address* is typically **`0x20`** for the *Motorola HC12* derivatives.
- The *Module base address* is typically **`0x28`** for the *Motorola HCS12* derivatives.

5.When those settings are done, any breakpoint which is set in *Non Volatile Memory* is considered by the debugger as an *Hardware Breakpoint*.

Figure 2.15 Hardware Breakpoint Configuration



If your application is loaded in RAM, breakpoints are software breakpoints. In this case the *Hardware Breakpoint* module gives you the possibility to debug with breakpoints and watchpoint (only one watchpoint is available).

NOTE	In Automatic mode, the *HC12* and *HCS12* hardware breakpoint modules allow only two breakpoints (or one watchpoint) at the same time. If you are debugging your code in FLASH, you can not set more than two breakpoints or one watchpoint. Some actions like “stepping over” or “stepping out” use one internal breakpoint and therefore reduce your amount of hardware breakpoint to one.

NOTE	If you are using a *Motorola HC12* derivative and would like to set hardware breakpoints in *Banked Memory Location*, set the *Continue on illegal break* check box.

WARNING!	The M68HC812A4 does not have any Hardware Breakpoint module.

3

ICD-12 Target Interface Manual

3.1 Introduction

An advanced feature of *Metrowerks debugger* for the embedded systems development world is the ability to load different Target Interfaces, which implements the interface with target systems. In this document, the specific features of the *ICD-12* Target Interface are described.

The *ICD-12* Target Interface allows to debug using *P&E Cable12* and *BDM-MULTILINK*. The *P&E Cable12* and *BDM-MULTILINK* are interfaces developed by *P&E Microcomputer Systems* and used by *the Metrowerks debugger* to communicate with a target system, based on a *Motorola HC12* or *HCS12*.

With this interface, you can download an executable program from the *Metrowerks CodeWarrior HC12 Studio*, to an external target system based on a *Motorola HC12* or *HCS12*, which will execute the program. You will also have the feedback of the real target system behavior to *Metrowerks debugger*.

The debugger will fully supervise and monitor the MCU of the target system i.e. control the CPU execution. You can read and write in internal/external memory when the MCU is in Background Mode. You have full control over the CPU state with the possibility to stop execution, to proceed in single step mode and to set breakpoints in the code.

NOTE	**Unconcerned Components** As the code is executed by an external processor, memory statistics are not available with the *ICD-12* Target Interface. Therefore, Profiling, Coverage analyzing and I/O simulation will not work with the *ICD-12* Target Interface.

For More Information: www.freescale.com

3.2 Interfacing Your System with the Target

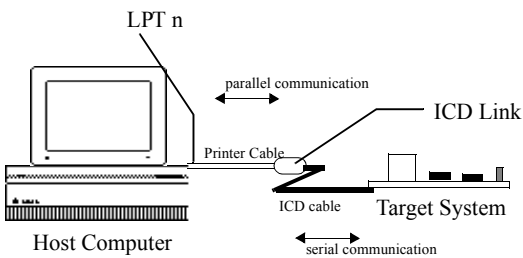
3.2.1 Hardware Connection

A parallel cable should be used for the communication between the *P&E Cable12* or *BDM-MULTILINK* and the host computer.

A parallel to serial interface is used for the communication between the target board and the host computer, as shown Figure 3.1.

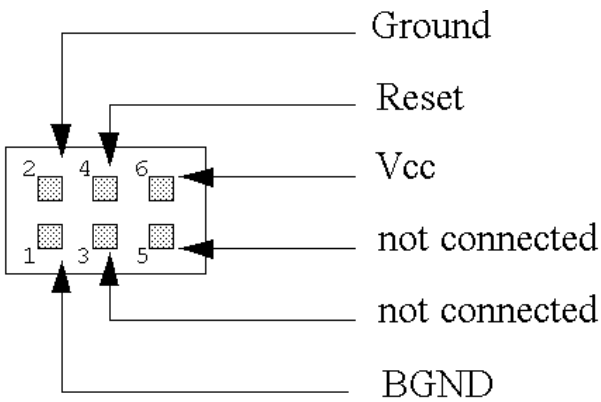
The communication to the target is serial. The communication protocol between the *P&E Cable12* or *BDM-MULTILINK* and host is fully handled by the *P&E unit_12z.dll* target driver which is automatically loaded with the *ICD-12* Target Interface.

Figure 3.1 Description of the hardware.



The target hardware has to be equipped with a BDM connector which connects the *P&E Cable12* or *BDM-MULTILINK*. The BDM port 6-pin connector is shown below (Figure 3.2).

Figure 3.2 The BDM port 6-pin connector.



The *P&E Cable12* or *BDM-MULTILINK* to target system communication is serial. The communication protocol is defined by Motorola in the *CPU 12 Reference Manual, section 8 (Development and Debug Support).* However the user does not need to know this protocol to run an *ICD-12* Target Interface debugging session.

The *P&E Cable12* or *BDM-MULTILINK* are power supplied by the target system. This power supply must be conformed to the TTL standard. If not, the *P&E Cable12* or *BDM-MULTILINK* should have its own supply. Please also refer to the *P&E Cable12* or *BDM-MULTILINK* hardware manual from *P&E.*

3.2.2 Loading the ICD-12 Target Interface

Usually the target is set in the **[HI-WAVE]** section of the PROJECT file, through the statement **Target=ICD12**.

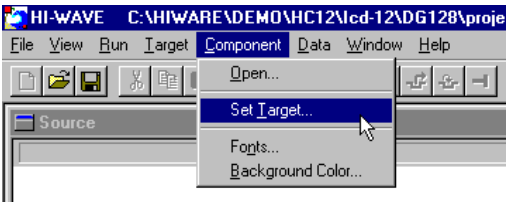
In this way, the *ICD-12* Target Interface detects automatically that the target is connected to your system.

However, if nothing is detected, the *Communication Device Specification* dialog pops up: the target is not connected or is connected to a different port.

If no target is set in the PROJECT file or if a different target is set, you can load the *ICD-12* Target Interface.

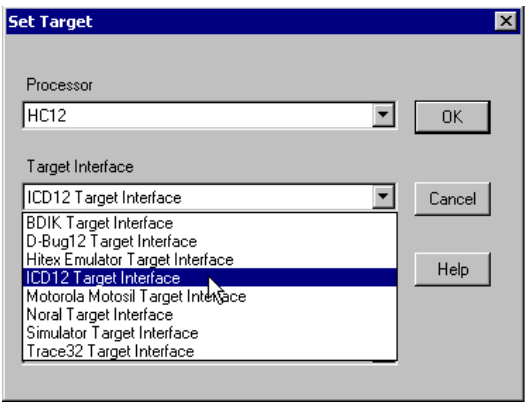
Select in the main menu **Component > Set Target...**, as shown in *Figure 3.3.*

Figure 3.3 Set Target dialog



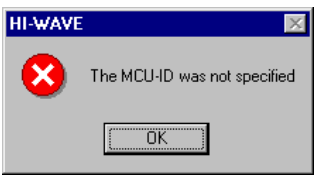
The *Set Target* dialog is displayed. Choose *ICD12 Target Interface* in the list of proposed targets and click *OK*.

Figure 3.4 List of available targets interfaces



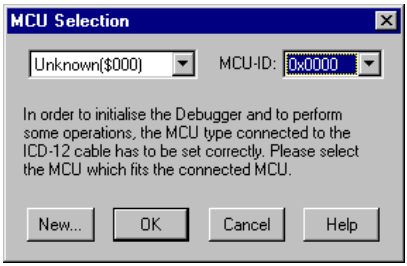
The debugger tries then to connect to the target: if the *ICD-12* Target Interface is used for the first time, the *MCU-ID* of the derivative must be entered. Figure 3.5 shows the dialog which is displayed.

Figure 3.5 MCU-ID is not specified



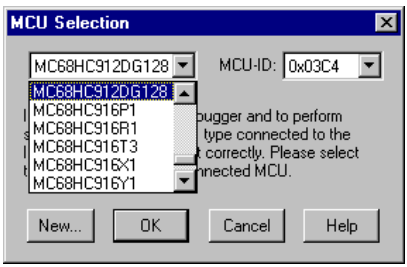
Click *OK*. The *MCU Selection* dialog is opened as shown in Figure 3.6.

Figure 3.6 MCU Selection dialog



In the drop down list box shown in Figure 3.7 select the derivative which is on your target system.

Figure 3.7 Drop down list box of the MCU Selection dialog

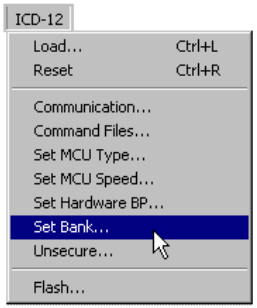


Close the *MCU Selection* dialog clicking *OK*, connection to target is established.

3.3 ICD-12 Target Interface Menu Entries

After loading the *ICD-12* Target Interface, the *Target* menu item is replaced by *ICD-12*. The different entries of the *ICD-12* menu are described in Figure 3.8:

Figure 3.8 ICD-12 Menu entries.



3.3.1 Load...

Choose **ICD-12 > Load...** to load the application to debug, i.e. e.g. a **.abs** file.

3.3.2 Reset

The menu entry **ICD-12 > Reset** executes the Reset Command File and resets the target system processor.

3.3.3 Communication... or Connect...

Select entry **ICD-12 > Communication...** or **ICD-12 > Connect...** to display the Communication Device Specification dialog. If the connection to the target has failed, the entry ***Communication...*** of menu *ICD-12* is replaced with ***Connect...*** .

3.3.4 Command Files

Select entry **ICD-12 > Command Files** to display the *ICD-12* Target Interface Command Files dialog.

3.3.5 Set MCU Type...

Choose **ICD-12 > Set MCU Type...** to open the MCU Selection dialog.

3.3.6 Set BDM Speed...

Choose **ICD-12 > Set BDM Speed...** to open the Set MCU Speed dialog.

3.3.7 Set Hardware BP...

This menu entry is available only if the connection with the target system has been established.

Choose **ICD-12 > Set Hardware BP...** to open the Hardware Breakpoint Configuration dialog.

3.3.8 Set Bank...

This menu entry is available only if the connection with the target system has been established.

Choose **ICD-12 > Set Bank...** to open the Banked Memory Location Dialog Box.

3.3.9 Unsecure...

This menu entry is available only if the connection with the target system has been established.

Choose **ICD-12 > Unsecure...** to open the Unsecure derivative dialog box.

3.3.10 Flash...

This menu entry is available only if the connection with the target system has been established.

Choose **ICD-12 > Flash...** to open the Non Volatile Memory Control Dialog.

3.4 ICD-12 Target Interface Dialogs

This section describes the dialogs which are specific to the *ICD-12* Target Interface.

Those dialogs are:

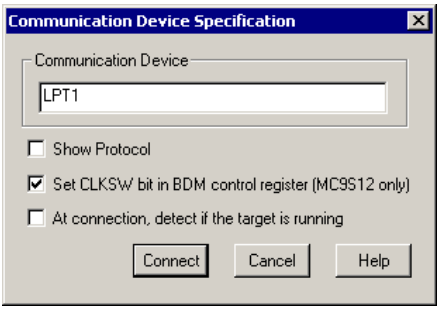
- the Communication Device Specification dialog,
- the *ICD-12* Target Interface Command Files dialog,
- the MCU Selection dialog,
- the Set MCU Speed dialog,
- the Hardware Breakpoint Configuration dialog,
- the Banked Memory Location Dialog Box,
- the Unsecure derivative dialog box,

- the Non Volatile Memory Control Dialog.

3.4.1 Communication Device Specification

Select entry **ICD-12 > Communication...** to display the *Communication Device Specification*.

Figure 3.9 Communication Device Specification dialog box.



If the connection to the target has failed, the entry *Communication...* of menu *ICD-12* is replaced with *Connect*. In this case it is possible to modify the parallel port used in the *Communication Device* edit box.

Click *OK or Connect* to try to connect to the target system through the *P&E Cable12* or *BDM-MULTILINK* connected to the parallel port specified in the *Communication Device* edit box.

When clicking *Cancel* in the *Communication Device Specification dialog,* the dialog box and the environment can be quit.

Communication Device: The *Communication Device* edit box allows to specify the current communication port. The default device is **`LPT1`**.

Show Protocol: The *Show Protocol* check box allows to switch on/off the displays of the messages sent between the *ICD-12* Target Interface and the debugger. If the *Show Protocol* box is checked, all the commands and responses sent and received are reported in the *Command Line* window.

Set CLKSW bit in BDM control register: With some *Motorola HCS12* derivatives (e.g. *MC9S12DP256* mask *0K79X*), when the PLL is enabled and the BDC is set to use the external oscillator frequency (this is the default, CLKSW=0), the BDC will no longer operate properly (chip defect). This problem did not affect the original *HC12* and *HCS12* devices. Solution: before the user code enables the PLL, the CLKSW bit

in the BDCSCR must be set. This can be done using this check box. We do not set this automatically because it may affect other devices.

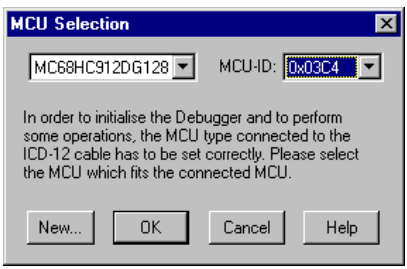
At Connection, detect if the target is running: At connection, the Target Interface can detect if the target processor is running. The target processor is not reset and can be halted later. Then application file (.ABS) symbols can be reloaded to match current code with initial application file and keep on debugging. This option can be enabled/ disabled (default) using this check box.

TIP	The Show Protocol is a useful debugging feature if there is a communication problem.

3.4.2 MCU Selection

Select entry **ICD-12 > Set MCU Type...** to display the *MCU Selection* dialog, as shown in Figure 3.10.

Figure 3.10 MCU Selection dialog



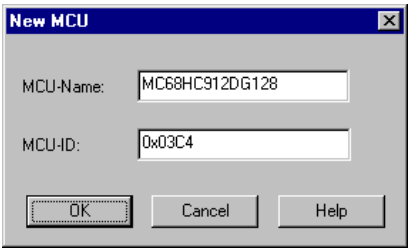
This dialog allows you to select the MCU currently used. There are two drop list Combo controls. They show the currently selected MCU name and the corresponding MCU-ID.

The information will be taken from the file **`MDSEMCU.INI`**. If a specific MCU is not found in this file, the user is advised to update his installation.

The selection will be saved and used as default for the next session.

If the expected MCU name is not contained in the MCU list, press the *New...* button, to open the following dialog (Figure 3.11):

Figure 3.11 Creation of a New MCU Name.

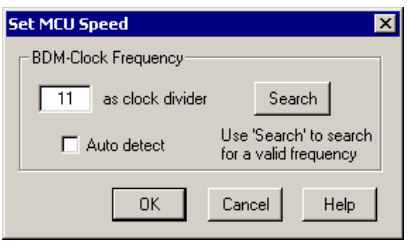


Through this dialog, the MCU name can be adapted. If the specified MCU-ID does not exist, a new entry is created in the **`MDSEMCU.INI`** file located in the windows directory of the host computer.

3.4.3 Set MCU Speed

Choose **ICD-12 > Set MCU Speed...** to open this dialog (Figure 3.12).

Figure 3.12 Set MCU Speed Dialog.



This dialog shows the current settings of the clock divider (also called IO_DELAY_COUNT) to be used by the MCU.

When *Ok* is clicked, the debugger tries to verify the communication with the set clock divider/IO_DELAY_COUNT.

Clock divider: The clock divider/IO_DELAY_COUNT has to be known by the *ICD-12* Target Interface for proper communication through the BDM. The clock divider/ IO_DELAY_COUNT used depend on the connected P&E Cable:

- for CABLE12: 16/(oscillator frequency in MHz)
- for CABLE12HS/BDM-MULTILINK: (200/(oscillator frequency in MHz)) -1

Search: If the *Search* button is used, the debugger automatically search for a valid clock divider/IO_DELAY_COUNT. The divider is then updated in the edit box. At start-up, the debugger will use the clock divider/IO_DELAY_COUNT stored in the PROJECT file through variable BDMClockSpeed.

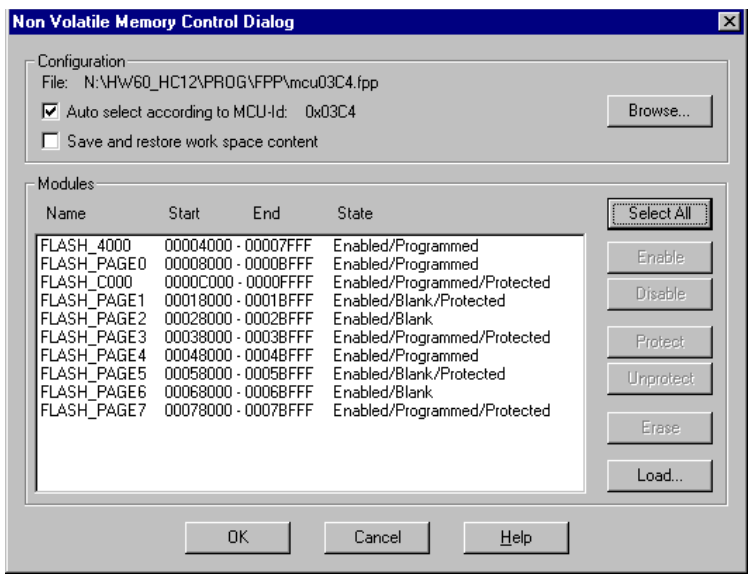
Auto detect: At start-up, if connection attempts fail and the check box *Auto detect* is checked, the debugger will try to find a fitting clock divider/IO_DELAY_COUNT value. If it was not possible to establish proper communication, the dialog is displayed.

NOTE	Selections are saved in the `PROJECT` file and used as default for the next session, through variables BDMClockSpeed and BDMAutoSpeed.

3.4.4 Non Volatile Memory Control Dialog

Select entry **ICD-12 > Flash...** to display the *Non Volatile Memory Control Dialog* dialog, as shown in Figure 3.13.

Figure 3.13 Non Volatile Memory Control Dialog.



The *ICD-12* Target Interface has a build-in *FLASH Programming* interface to program on-chip *FLASH EEPROM* modules, that you can open when choosing **ICD-12 > Flash...** entry in the *ICD-12* menu. Please see the *Flash Programming* manual to know more about the usage of this interface.

NOTE	This FLASH Programming utility must be licensed to work without size limitation. Check your license configuration.

3.5 Status Bar Information for the ICD-12 Target Interface

When the *ICD-12* Target Interface has been loaded, specific information are given in the debugger's status bar (see Figure 3.14)

Figure 3.14 Status bar

From left to right, the name of the selected MCU and the status (target status) are displayed.

3.5.1 Status Messages

3.5.1.1 ICD-12 ready

Metrowerks debugger is ready and waits until a new target or application is loaded. This message is generated once the *Metrowerks debugger* has been started and the connection to target system has been established.

3.5.1.2 No Link To Target

Connection to the target system has failed.

3.5.1.3 RUNNING

The application is currently executing on the target.

3.5.1.4 HALTED

Execution of the application has been stopped on user request. The menu entry **Run > Halt** or the *Halt* icon in the tool bar has been selected.

3.5.1.5 RESET

This message is generated when the *Metrowerks debugger* has been reset on user request. The menu entry **ICD-12 > Reset** or the *Reset* icon in the tool bar has been selected, or the command Reset has been used.

3.5.2 Stepping and Breakpoint Messages

3.5.2.1 STEPPED

Execution of the application has been stopped after a single step on source level. The menu entry **Run> Single Step** or the *Single Step* icon in the tool bar has been selected.

3.5.2.2 STEPPED OVER

Execution of the application has been stopped after a step over a function call. The menu entry **Run > Step Over** or the *Step Over* icon in the tool bar has been selected.

3.5.2.3 STOPPED

Execution of the application has been stopped after a step out from function call. The menu entry **Run> Step Out** or the *Step Out* icon in the tool bar has been selected.

3.5.2.4 TRACED

Execution of the application has been stopped after an single step on assembler level. The menu entry **Run > Assembly Step** or the *Assembly Step* icon in the tool bar has been selected.

3.5.2.5 BREAKPOINT

Execution of the application has been stopped because a breakpoint has been reached.

3.5.2.6 WATCHPOINT

Execution of the application has been stopped because a watchpoint has been reached.

3.6 ICD-12 Target Interface Default Environment

3.6.1 Default Target Setup

As any other target, the *ICD-12* Target Interface can be loaded from the *Target* menu or can be set as a default target in the `PROJECT` file which should be located in the working directory.

Typically the target is set in the **`[HI-WAVE]`** section from the `PROJECT` file as shown above. However, if the target is not defined, yo can load the *ICD-12* Target Interface interactively. Please refer to section *Loading the ICD-12 Target Interface* of this manual.

Listing 3.1 Example of `[HI-WAVE]` section from `PROJECT` file:

```
[HI-WAVE]
Window0=Source      0    0  60  30
Window1=Assembly   60    0  40  30
Window2=Procedur    0   30  60  25
Window3=Register   60   30  40  30
Window4=Memory     60   60  40  40
Window5=Data        0   55  60  23
Window6=Data        0   78  60  22
Target=ICD12
```

NOTE Please see the Manual Engine HC12.pdf for further information about the `PROJECT` file.

3.6.2 ICD-12 Target Interface Environment Variables

This section describes the environment variables which are used by the *ICD-12* Target Interface.

The *ICD-12* Target Interface specific environment variables are:

- BDMAutoSpeed
- BDMClockSpeed
- COMDEV

- DETECTRUNNING
- MCUID
- SETCLKSW
- SHOWPROT

Some other common Target Interface features are using variables. Those variables are:

- BANKWINDOWn
- CHIPSECURE
- CMDFILEn
- HWBPD_MCUIDnnn_BKPT_REMAPn
- HWBPMn

These variables are stored in the **[ICD12]** section from the PROJECT file.

Listing 3.2 Example of [ICD12] section from PROJECT file:

```
[ICD12]
CMDFILE0=CMDFILE STARTUP ON ".\cmd\p&e_startup.cmd"
CMDFILE1=CMDFILE RESET ON ".\cmd\p&e_reset.cmd"
CMDFILE2=CMDFILE PRELOAD ON ".\cmd\p&e_preload.cmd"
CMDFILE3=CMDFILE POSTLOAD ON ".\cmd\p&e_postload.cmd"
CMDFILE4=CMDFILE VPPON ON ".\cmd\p&e_vppon.cmd"
CMDFILE5=CMDFILE VPPOFF ON ".\cmd\p&e_vppoff.cmd"
CMDFILE6=CMDFILE UNSECURE OFF ".\cmd\p&e_erase_unsecure_hcs12.cmd"
BANKWINDOW0=BANKWINDOW PPAGE ON 0x8000..0xBFFF 0x30 64
BANKWINDOW1=BANKWINDOW DPAGE OFF 0x7000..0x7FFF 0x34 256
BANKWINDOW2=BANKWINDOW EPAGE OFF 0x400..0x7FF 0x36 256
CHIPSECURE=CHIPSECURE SETUP 0xFF0F 0x3 0x2
MCUId=0x3C6
SETCLKSW=1
BDMClockSpeed=11
HWBPM0=HWBPM MODE AUTOMATIC BPM22BITS 0x28 SKIP_OFF
HWBPD_MCUID3C6_BKPT_REMAP0=HWBPM REMAP_22BITS RANGE 0x4000 0x7FFF 0x3E
HWBPD_MCUID3C6_BKPT_REMAP1=HWBPM REMAP_22BITS RANGE 0xC000 0xFFFF 0x3F
HWBPM1=HWBPM SET16BITS 0x0 0x0 0x0 0x0
HWBPM2=HWBPM SET22BITS 0x0 0x0 0x0 0x0
BDMAutoSpeed=1
```

NOTE	Please see the Flash Programming manual for NV_PARAMETER_FILE, NV_SAVE_WSP and NV_AUTO_ID variables description.

BDMAutoSpeed

Short Description

Set if the debugger must check automatically for BDM frequency at connection

Syntax

```
BDMAutoSpeed= 0 | 1
```

Alias

None

File

`PROJECT file`

Section

`[ICD12]`

Components

ICD-12 Target Interface.

Description

If the **`BDMAutoSpeed`** variable is set to 1, the debugger automatically searches for the correct BDM frequency at connection.

This variable is set according to the *Auto Detect* check box in the Set MCU Speed dialog.

For example,

```
BDMAutoSpeed=1
```

BDMClockSpeed

Short Description

Divider used to calculate the BDM frequency

Syntax

```
BDMCLockSpeed=<frequency divider>
```

Alias

None

File

`PROJECT file`

Section

`[ICD12]`

Components

ICD-12 Target Interface.

Description

The **`BDMClockSpeed`** contains a divider which allows the *P&E Cable12* or *BDM-MULTILINK* to calculate the BDM frequency.

This variable is set according to the value entered in the edit box in the Set MCU Speed dialog.

For Example,

```
BDMClockSpeed=5
```

COMDEV

Short Description

Communication device

Syntax

```
COMDEV=LPTn
```

Alias

None

File

`PROJECT file`

Section

[ICD12]

Components

ICD-12 Target Interface.

Description

The communication port to be used on the host computer can be specified using the variable **`COMDEV`**.

`LPT1` is the default communication device.

This variable is set according to the *Communication Device* edit box of the Communication Device Specification dialog.

For example,

```
COMDEV=LPT1
```

DETECTRUNNING

Short Description

Running target at connection detection on/off.

Syntax

```
DETECTRUNNING=0|1
```

Alias

None

File

```
PROJECT file
```

Section

```
[ICD12]
```

Components

ICD-12 Target Interface.

Description

The **DETECTRUNNING** variable is used to store the status of the mechanism which detects if the derivative connected through the *P&E Cable12* or *BDM-MULTILINK* cable is running at connection time.

This variable is set according to the *At Connection, detect if the target is running* check box in the Communication Device Specification dialog.

For example,

```
DETECTRUNNING=1
```


For More Information: www.freescale.com

MCUID

Short Description

MCU-ID of the connected derivative

Syntax

```
MCUID=<MCU-ID of the connected derivative>
```

Alias

None

File

`PROJECT file`

Section

`[ICD12]`

Components

ICD-12 Target Interface.

Description

The **`MCUID`** variable is used to store the MCU-ID of the derivative connected through the *P&E Cable12* or *BDM-MULTILINK* cable.

Each MCU has its own MCU-ID which is assigned by *Motorola*.

This variable is set according to the MCU-ID selected in the MCU Selection dialog.

For example,

```
MCUID=0x3c4
```

SETCLKSW

Short Description

CLKSW on/off.

Syntax

```
SETCLKSW=0|1
```

Alias

None

File

`PROJECT file`

Section

`[ICD12]`

Components

ICD-12 Target Interface.

Description

The **SETCLKSW** variable is used to store the status of the mechanism which uses the CLKSW bits of the BDCSCR register to synchronize the communication.

This variable is set according to the *Set CLKSW bit in BDM control register* check box in the Communication Device Specification dialog.

For example,

```
SETCLKSW=1
```

SHOWPROT

Short Description

Set *Show Protocol* On/Off

Syntax

```
SHOWPROT=1|0
```

Alias

None

File

`PROJECT file`

Section

`[ICD12]`

Components

ICD-12 Target Interface.

Description

If the *Show Protocol* is used, all the commands and responses sent and received are reported in the *Command Line* component of the debugger.

If the variable is set to 1, *Show Protocol* is activated.

This variable is set according to the *Show Protocol* check box of the Communication Device Specification dialog. For example,

```
SHOWPROT=1
```

TIP	The Show Protocol is a useful debugging feature if there is a communication problem.

3.7 ICD-12 Target Interface Command Line commands

This section describes the *ICD-12* Target Interface specific commands which can be used when the *ICD-12* Target Interface is set.

The *ICD-12* Target Interface specific commands are:

- BDMAREASUPPORT
- HWBREAKONLY
- LISTALLMCUIDS
- PROTOCOL
- REGBLKADR
- RESET

Some other common Target Interface features are using commands. Those commands are:

- BANKWINDOW
- CHIPSECURE
- HWBPM
- CMDFILE

Those commands can be entered in the file or in the *Command Line* component of the debugger.

BDMAREASUPPORT

Short Description

step in BDM ROM

Syntax

BDMAREASUPPORT STATE | OFF | ON [<first address > <last address >]

Alias

None

Components

ICD-12 Target Interface.

Description

This command allows to step in BDM ROM range. Typically not useful, as no program code should be stored in this area.

HWBREAKONLY

Short Description

force the debugger to use only hardware breakpoints

Syntax

HWBREAKONLY OFF | ON | STATE

Alias

None

Components

ICD-12 Target Interface.

Description

By default the debugger first tries to set a software breakpoint, then if it is not possible it tries to set a hardware breakpoint.

Using this command, you can disable this mechanism and force the debugger to use only hardware breakpoints without attempting to use first SWI software breakpoint.

LISTALLMCUIDS

Short Description

to list all the MCU-IDs

Syntax

LISTALLMCUIDS OFF | ON | STATE

Alias

None

Components

ICD-12 Target Interface.

Description

This command has been added, as MCU Selection dialog lists by default only the *HC12* and *HCS12* derivatives.

LISTALLMCUIDS allows to display all the defined MCU-IDs.

PROTOCOL

Short Description

switch on/off the *Show Protocol* functionality

Syntax

```
PROTOCOL ON|OFF
```

Alias

None

Components

ICD-12 Target Interface.

Description

If this command is used, all the messages sent to and received from the *ICD-12* interface are reported in the *Command Line* window of the debugger.

For Example

```
PROTOCOL ON
```

TIP	The Show Protocol is a useful debugging feature if there is a communication problem.

REGBLKADR

Short Description

register block moving handling

Syntax

REGBLKADR <new register block address>

Alias

None

Components

ICD-12 Target Interface.

Description

On *HCS12* derivatives, the Target Interface can automatically track device registers block ($0-$3FF) relocated via INITRG register setup.

For *HC12* derivatives, a command has been implemented to specify the new location of the register block: REGBLKADR

- At connection the register block address is 0
- When a reset command is issued, the register block address is set to 0
- When REGBLKADR is executed, the information is passed to the target and the debugger is updated.

This command must be performed after the register block address has been changed by the user application (when the application is stopped after a step, a stop, a breakpoint ... or any other event).

RESET

Short Description

reset of the target board

Syntax

RESET

Alias

None

Components

ICD-12 Target Interface.

Description

With this command it is possible to reset the target from the *Command Line* component of the debugger.

The target board is reset and the reset vector is read in the target memory from the vector table (address **`0xFFFE`**).

For example,

```
RESET
```

4

HC12 and HCS12 Banked Memory support

The PPAGE, DPAGE and the EPAGE banked memory location are supported by the *Metrowerks debugger* Target Interfaces, depending on the debugged application and on the debugged *HC12/HCS12* derivative.

4.1 Banked Memory Location Dialog Box

The *Banked Memory Location* dialog box is available only if the connected derivative is a *Motorola HC12 (CPU12)* or *HCS12*.

The *Banked Memory Location* dialog box can be opened by selecting the menu entry **"TargetName" > Set Bank...**. (In this section, **TargetName** is the name of the Target Interface, like *SDI*, *Hitex*, *BDIK*, *ICD-12*, *Noral-BDM*, etc.) Using some Target Interfaces, the *Banked Memory Location* dialog box automatically pops up when the Target Interface is used with a *Motorola HC12* or *HCS12* derivative that supports banking. In this case, it also pops up when the banked memory area locations are not defined in the project file of the current project directory.

In this dialog box you can define which banked memory you want to use and its location. The PPAGE, DPAGE and the EPAGE indexes are supported, if they are available on the currently connected *HC12* or *HCS12* derivative.

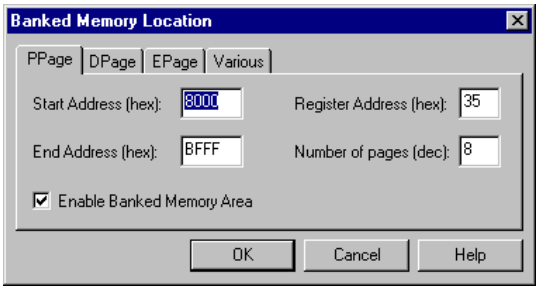
The *Banked Memory Location* dialog box can have up to four index tabs:

- The PPAGE index tab,
- The DPAGE index tab,
- The EPAGE index tab,
- The Various index tab (not all Target Interfaces).

4.1.1 PPAGE index tab

The PPAGE index tab of the *Banked Memory Location* dialog box lets you set up the PPAGE banked memory area.

Figure 4.1 Banked Memory Location dialog (PPAGE index tab)



Once you have enabled PPAGE memory banking by checking the *Enable Banked Memory Area* check box, you must set the start address and the end address of this memory range.

The PPAGE register address must be specified in hexadecimal (e.g. 0x35 for HC812A4, 0xFF for HC912DG128, 0x30 for MC9S12DP256B).

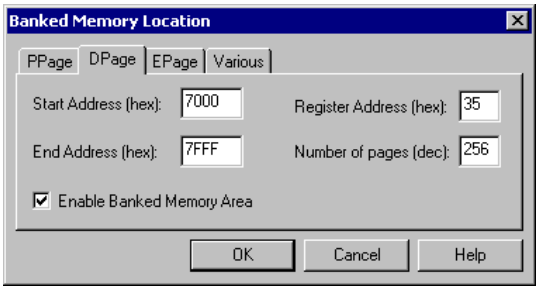
The number of pages must be specified in decimal (e.g. 0 to 256 for HC812A4, 8 for HC912DG128, 64 for the MC9S12DP256B).

NOTE For the *Hitex* Target Interface, the PPAGE index tab does not appear in this dialog box if the PPAGE register is not available on the currently connected *Motorola HC12* derivative. For this Target Interface it is not needed to enter the PPAGE register address.

4.1.2 DPAGE index tab

The DPAGE index tab of the *Banked Memory Location* dialog box lets you set up the DPAGE banked memory area.

Figure 4.2 Banked Memory Location dialog (DPAGE index tab)



Once you have enabled DPAGE memory banking by checking the *Enable Banked Memory Area* check box, you must set the start address and the end address of this memory range.

The number of pages must be specified in decimal (e.g. 0 to 256 for HC812A4).

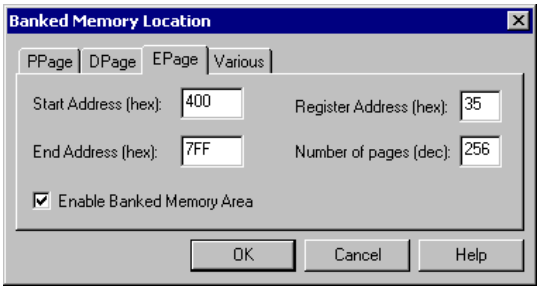
The DPAGE register address must be specified in hexadecimal (e.g. 0x34 for HC812A4).

NOTE	For the *Hitex* Target Interface, the DPAGE index tab does not appear in this dialog box if the DPAGE register is not available on the currently connected *Motorola HC12* derivative. For this Target Interface it is not needed to enter the DPAGE register address.

4.1.3 EPAGE index tab

The EPAGE index tab of the *Banked Memory Location* dialog box lets you set up the EPAGE banked memory area.

Figure 4.3 Banked Memory Location dialog (EPAGE index tab)



Once you have enabled EPAGE memory banking by checking the Enable Banked Memory Area check box, you must set the start address and the end address of this memory range.

The number of pages must be specified in decimal (e.g. 0 to 256 for HC812A4).

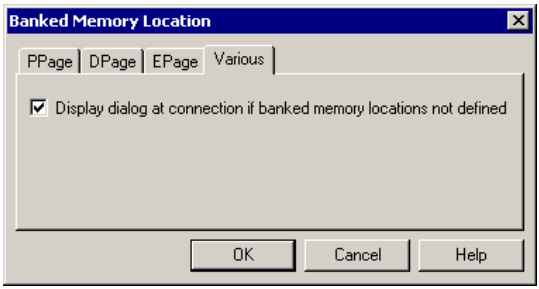
The EPAGE register address must be specified in hexadecimal (e.g. 0x36 for HC812A4).

NOTE For the *Hitex* Target Interface, the EPAGE index tab does not appear in this dialog box if the EPAGE register is not available on the currently connected *Motorola HC12* derivative. For this Target Interface it is not needed to enter the EPAGE register address.

4.1.4 Various index tab (not all Target Interfaces)

The Various index tab of the *Banked Memory Location* dialog box is not available for all Target Interfaces.

Figure 4.4



If you are using an *HC12* derivative which supports banking and you don't want to enable this mechanism, or if you want to use only one bank out of three, you can suppress the automatic display of the *Banked Memory Location* dialog by checking the *Display dialog at connection if banked memory locations not defined* check box.

NOTE	The settings entered in this dialog box are stored for a later debugging session in the ["targetName"] section of the project file.

NOTE	When using the *Hitex* Target Interface and the M68HC12DG128 DProbeHC12-DG, at least one page must be defined from 0x8000 to 0xBFFF. Otherwise, some display problems might be encountered in the Memory component of the *Metrowerks debugger*.

4.2 Associated Commands

The following sections describe the Banked Memory Location Command Line commands which are used by the Target Interface. These variables are:

BANKWINDOW

Those commands can be entered in the Target Interface associated command files or in the Command Line component of the debugger.

For More Information: www.freescale.com

The Banked Memory Location commands which are used by the Target Interface are described as shown in the following table.

Topic	Description
Short Description	Provides a short description of the command.
Syntax	Specifies the syntax of the command in a EBNF format.
Description	Provides a detailed description of the command and how to use it.
Example	Small example of how to use the command.

The following sections describe each command related to the Banked Memory Location available for the Target Interface. The variables are listed in alphabetical order.

BANKWINDOW

Short Description

Specify a banked memory area and its status (enable/disable).

Syntax

```
BANKWINDOW <bank> [OFF|ON] [<range> <reg> <numofpages>]
with
bank = (PPAGE | DPAGE | EPAGE)
or
BANKWINDOW VARIOUS [DLGATCONNECT|NODLGATCONNECT]
```

Description

The command BANKWINDOW allows to set up the debugger to work in banked memory model.

Three different Banked Memory Area can be defined: DPAGE, EPAGE and PPAGE. Each banked memory area has an associated bank register, which is displayed in the *Register* component of the *Metrowerks debugger*.

Using BANWINDOW PPAGE ... command will have the same effect than using the PPAGE index tab in the Banked Memory Location Dialog Box.

Using BANWINDOW DPAGE ... command will have the same effect than using the DPAGE index tab in the Banked Memory Location Dialog Box.

Using BANWINDOW EPAGE ... command will have the same effect than using the EPAGE index tab in the Banked Memory Location Dialog Box.

Using BANWINDOW VARIOUS ... command will have the same effect than using the Various index tab in the Banked Memory Location Dialog Box.

A banked memory area is defined by its start address, end address and the address of the Bank register.

The maximum number of pages parameter allows to see in the memory component only the available pages.

The status of the banking mechanism in the debugger is also monitored through this command: a command can be defined, but the debugger banking mechanism can be disabled.

Consider the command:

```
BANKWINDOW PPAGE ON 0x8000..0xBFFF 0x30 64
```

This command allows to use the banked memory model in the debugger using the MC9S12DP256B.

This commands means the PPAGE register located at address 0x30 must be used to build the PC address when the code is located in banked memory area, from 0x8000 to 0xBFFF. The 64 first page in the memory map are visible (page 0x3F is the last one).

The PPAGE register (located at address 0x30) will be displayed in the register component.

The bank settings are stored in the ["targetName"] section of the PROJECT file using variable BANKWINDOWn.

Example

The bank memory area status can be get typing BANKWINDOW without any parameters in the Command Line component.

```
in>bankwindow
PPAGE Settings:
Status: enabled
Reg. Adr: 0x30
Range: 0x8000 to 0xbfff
Number of Pages: 64
```


For More Information: www.freescale.com

```
DPAGE Settings:
Status: disabled
Reg. Adr: 0x34
Range: 0x7000 to 0x7fff
Number of Pages: 0

EPAGE Settings:
Status: disabled
Reg. Adr: 0x36
Range: 0x400 to 0x7ff
Number of Pages: 0
in>
```

The status of the PPAGE Banked Memory area can be changed:

```
in>BANKWINDOW PPAGE OFF
in>BANKWINDOW
PPAGE Settings:
Status: disabled
Reg. Adr: 0x30
Range: 0x8000 to 0xbfff
Number of Pages: 64

DPAGE Settings:
Status: disabled
Reg. Adr: 0x34
Range: 0x7000 to 0x7fff
Number of Pages: 0

EPAGE Settings:
Status: disabled
Reg. Adr: 0x36
Range: 0x400 to 0x7ff
Number of Pages: 0
in>
```

4.3 Associated Environment Variables

The following sections describe the Banked Memory Location environment variables which are used by the Target Interface. These variables are:

BANKWINDOWn

These variables are stored in the ["targetName"] section from the project file.

Example of the [BDIK] target section from a project file:

```
[BDIK]
BANKWINDOW0=BANKWINDOW PPAGE ON 0x8000..0xBFFF 0x30 64
BANKWINDOW1=BANKWINDOW DPAGE OFF 0x7000..0x7FFF 0x34 256
BANKWINDOW2=BANKWINDOW EPAGE OFF 0x400..0x7FF 0x36 256
```

The Banked Memory Location environment variables which are used by the Target Interface are described as shown in the following table.

Topic	Description
Short Description	Provides a short description of the variable.
Syntax	Specifies the syntax of the variable in a EBNF format.
Default	Shows the default setting for the variable.
Description	Provides a detailed description of the variable and how to use it.
Example	Small example of how to use the variable.

The following sections describe each variable available for the Target Interface. The variables are listed in alphabetical order.

BANKWINDOWn

Short Description

Contains a BANKWINDOW Command Line command to be used to set up the Banked Memory support.

Syntax

```
BANKWINDOWn=<one BANKWINDOW Command Line command>
```

Default

All available banked memory area are disabled by default.

The default PPAGE memory banked area is 0x8000 to 0xBFFF, 8 pages allowed, with PPAGE register at address 0x35.

The default DPAGE memory banked area is 0x7000 to 0x7FFF, 256 pages allowed, with PPAGE register at address 0x34.

The default EPAGE memory banked area is 0x400 to 0x7FF, 256 pages allowed, with PPAGE register at address 0x36.

The default settings for the VARIOUS page is that the *Banked Memory Location* dialog is displayed automatically when connecting when settings are not done (do only apply to the *Hitex* Target Interface).

Description

The BANKWINDOWn variable specifies a command file definition using BANKWINDOW Command Line command. Three or four of those entries should be present in the `PROJECT` file, depending on the Target Interface.

Those variables are used to store the Banked Memory Location definition (range, address, number of pages) and status (enable/disable) specified either with the BANKWINDOW Command Line command or through the Banked Memory Location Dialog Box.

Example

```
BANKWINDOW0=BANKWINDOW PPAGE OFF 0x8000..0xBFFF 0x30 64
BANKWINDOW1=BANKWINDOW DPAGE OFF 0x7000..0x7FFF 0x34 256
BANKWINDOW2=BANKWINDOW EPAGE OFF 0x400..0x7FF 0x36 256
BANKWINDOW3=BANKWINDOW VARIOUS DLGATCONNECT
```

5

Unsecure Motorola HCS12 derivatives

Motorola HCS12 derivatives include a security circuitry to prevent unauthorized access to contents of FLASH, EEPROM and RAM memory when background debugging.

The *ICD-12* Target interface provides a "Unsecure" function.

The **ICD-12 > Unsecure** menu command (and corresponding command line command CHIPSECURE UNSECURE) allows to connect to the target through the Unsecure derivative dialog box and to execute the Unsecure Command File in order to unsecure the connected derivative.

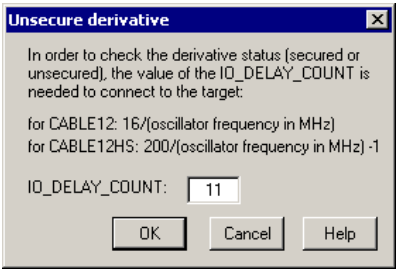
NOTE	Some of the *Motorola HCS12* derivatives can NOT be unsecured while in Special mode (this is not possible with all MC9S12DP256 derivatives mask, please check *Motorola User Manuals* for the connected derivative).

5.1 Unsecure derivative dialog box

When selecting **ICD-12 > Unsecure**, the *Unsecure derivative* dialog is displayed.

For More Information: www.freescale.com

Figure 5.1 Unsecure derivative dialog



In order to connect to the derivative, the correct IO_DELAY_COUNT must be entered. The *Unsecure derivative* dialog offers to connect using the last used IO_DELAY_COUNT, please check the value:

For Cable12:

IO_DELAY_COUNT = 16/(oscillator frequency in MHz)

For Cable12HS/BDM Multilink:

IO_DELAY_COUNT = (200 / (oscillator frequency in MHz)) - 1

CAUTION	The IO_DELAY_COUNT must be calculated. It is not possible for the debugger to detect automatically the IO_DELAY_COUNT when the chip is secured.

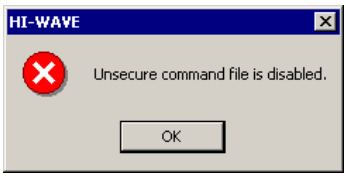
Once the correct IO_DELAY_COUNT has been entered, click *OK* to start the unsecure process.

The debugger will then attempt to connect to the target using this specified IO_DELAY_COUNT.

If basic connection is established, the Unsecure Command File is executed.

CAUTION	In case the Unsecure Command File has not been set up in the Target Interface Command Files dialog, the following dialog box is displayed:

Figure 5.2 Unsecure command file disabled dialog box



The unsecure process checks the security byte to see if the device is unsecured, according to a mask and a compare value: if (((value in security byte)& mask) == compare value)) then the chip is secured.

NOTE	The address of the security register, the mask and the compare value can be modified using command CHIPSECURE SETUP (those parameters are then stored in the project file.).

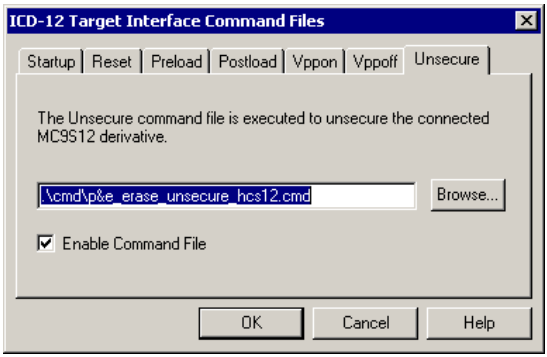
5.2 Unsecure Command File

The *Unsecure* command file can be set up using the *ICD-12 Target Interface Command Files* dialog. Choose **ICD-12 > Command Files** and click the *Unsecure* index tab.

This command file is executed in order to unsecure a secured *Motorola HCS12* derivative (using **ICD-12 > Unsecure** menu entry).

For More Information: www.freescale.com

Figure 5.3 Unsecure index tab



Here is an example of command file to be used to unsecure an *HCS12* derivative:

```
// HCS12 Core erasing + unsecuring command file:
// These commands mass erase the chip then program the
// security byte to 0xFE (unsecured state).
// Evaluate the clock divider to set
// in ECLKDIV/FCLKDIV registers:
// An average programming clock of 175 kHz is chosen.
// If the oscillator frequency is less than 10 MHz,
// the value to store in ECLKDIV/FCLKDIV is equal to
// " oscillator frequency(kHz) / 175 ".

// If the oscillator frequency is higher than 10 MHz,
// the value to store
// in ECLKDIV/FCLKDIV is equal to
// " oscillator frequency (kHz) / 1400  + 0x40
// (to set PRDIV8 flag)".

// Datasheet proposed values:
//
// oscillator frequency
// ECLKDIV/FCLKDIV value (hexadecimal)
//
//  16 MHz            $49
//   8 MHz            $27
//   4 MHz            $13
//   2 MHz             $9
//   1 MHz             $4
```

```
define CLKDIV 0x49

FLASH MEMUNMAP   // do not interact with regular flash
programming monitor

//mass erase flash
wb 0x100 CLKDIV  // set FCLKDIV clock divider
wb 0x103 0       // FCFNG select block 0
wb 0x102 0x10    // set the WRALL bit in FTSTMOD
                 // to affect all blocks
wb 0x104 0xFF    // FPROT all protection disabled
wb 0x105 0x30    // clear PVIOL and ACCERR in FSTAT register
ww 0x108 0xD000 // write to FADDR address register
ww 0x10A 0x0000 // write to FDATA data register
wb 0x106 0x41    // write MASS ERASE command in FCMD register
wb 0x105 0x80    // clear CBEIF in FSTAT register
                 //to execute the command
wait 20          // wait for command to complete

//mass erase eeprom
wb 0x110 CLKDIV  // set ECLKDV clock divider
wb 0x114 0xFF    // EPROT all protection disabled
wb 0x115 0x30    // clear PVIOL and ACCERR in ESTAT register
ww 0x118 0x0400  // write to EADDR eeprom address register
ww 0x11A 0x0000  // write to EDATA eeprom data register
wb 0x116 0x41    // write MASS ERASE command in ECMD register
wb 0x115 0x80    // clear CBEIF in ESTAT register
                 // to execute the command
wait 20          // wait for command to complete

reset

//reprogram Security byte to Unsecure state
wb 0x100 CLKDIV  // set FCLKDIV clock divider
wb 0x103 0       // FCFNG select block 0
wb 0x104 0xFF    // FPROT all protection disabled
wb 0x105 0x30    // clear PVIOL and ACCERR in FSTAT register
ww 0xFF0E 0xFFFE // write security byte to "Unsecured" state
wb 0x106 0x20    // write MEMORY PROGRAM command
                 // in FCMD register
wb 0x105 0x80    // clear CBEIF in FSTAT register
                 // to execute the command
wait 20          // wait for command to complete

reset
```

```
FLASH MEMMAP     // restore regular flash programming monitor
undef CLKDIV      // undefine variable
```

5.3 Associated Commands

The following sections describe the HCS12 Unsecure Command Line commands which are used by the Target Interface. These variables are:

CHIPSECURE

Those commands can be entered in the Target Interface associated command files or in the Command Line component of the debugger.

The HCS12 Unsecure commands which are used by the Target Interface are described as shown in the following table.

Topic	Description
Short Description	Provides a short description of the command.
Syntax	Specifies the syntax of the command in a EBNF format.
Description	Provides a detailed description of the command and how to use it.
Example	Small example of how to use the command.

The following sections describe each command related to the Banked Memory Location available for the Target Interface. The variables are listed in alphabetical order.

CHIPSECURE

Short Description

Setup the HCS12 Unsecure mechanism and unsecure the connected derivative.

Syntax

```
CHIPSECURE UNSECURE
CHIPSECURE SETUP <addr. reg to check> <mask> <compare value>
```

Description

The command CHIPSECURE SETUP allows to set up the debugger unsecure mechanism.

<addr. reg to check> : address of the security register (by default 0xFF0F)

<mask> : comparison mask for the security register (by default 0x03)

<compare value> : comparison value for the security register (by default 0x02)

The command CHIPSECURE UNSECURE allows to unsecure the connected derivative (same than selecting **ICD-12 > Unsecure**), using the settings performed using the CHIPSECURE SETUP command.

When using CHIPSECURE UNSECURE, the Unsecure command file is executed and the secured derivative check process is performed: to check if the derivative is unsecured, the debugger reads **<addr. reg to check>**, mask it with **<mask>** and compare it to **<compare value>**.

Example

The following command setup the CHIPSECURE for most HCS12 derivatives:

```
in>CHIPSECURE SETUP 0xFF0F 0x3 0x2
in>
```

5.4 Associated Environment Variables

The following sections describe the HC12 Unsecure environment variables which are used by the Target Interface. These variables are:

CHIPSECURE

These variables are stored in the ["targetName"] section from the project file.

Example of the [ICD12] target section from a project file:

```
[ICD12]
CHIPSECURE=CHIPSECURE SETUP 0xFF0F 0x3 0x2
```

The HCS12 Unsecure environment variables which are used by the Target Interface are described as shown in the following table.

Topic	Description
Short Description	Provides a short description of the variable.
Syntax	Specifies the syntax of the variable in a EBNF format.
Default	Shows the default setting for the variable.
Description	Provides a detailed description of the variable and how to use it.
Example	Small example of how to use the variable.

The following sections describe each variable available for the Target Interface. The variables are listed in alphabetical order.

CHIPSECURE

Short Description

Contains a CHIPSECURE Command Line command to be used to set up the HCS12 Unsecure mechanism.

Syntax

```
CHIPSECURE=<CHIPSECURE SETUP Command Line command>
```

Description

The CHIPSECURE variable specifies the HCS12 Unsecure mechanism setup using a CHIPSECURE Command Line command.

Example

```
CHIPSECURE=CHIPSECURE SETUP 0xFF0F 0x3 0x2
```

6

Target Interface Commands Files

6.1 Target Interface Associated Command Files

The Target Interfaces offer the possibility to play a specific command file on different events:

- at connection: Startup Command File,
- at reset: Reset Command File,
- right before a file is loaded: Preload Command File,
- right after a file has been loaded: Postload Command File.
- right before a "Non Volatile Memory" is erased or right before a file is programmed in "Non Volatile Memory": Vppon Command File. This command file can be used for example to enable a programming voltage by software. This command file is not available for all Target Interfaces.
- right after a "Non Volatile Memory" has been erased or right after a file has been programmed in "Non Volatile Memory": Vppoff Command File. This command file can be used for example to disable a programming voltage by software. This command file is not available for all Target Interfaces.

The command files full name and status (enable/disable) can be specified either with the CMDFILE Command Line command or using the Command Files dialog.

You can use any *Metrowerks debugger* command in those files and take advantage of the wide set of commands introduced in the *Metrowerks debugger* manual to setup the target hardware on one of those events.

For More Information: www.freescale.com

Listing 6.1 Example of a command file content

```
WB 0x0035 0x00
WB 0x0012 0x11
PROTOCOL OFF
```

The `WB 0x0035 0x00` command sets memory location 0x35 to 0.

The `WB 0x0012 0x11` command sets memory location 0x12 to 0x11.

The command `PROTOCOL OFF` switch of the Show Protocol.

6.1.1 Startup Command File

The Startup command file is executed by the *Metrowerks debugger* straight after the Target Interface has been loaded.

The Startup command file full name and status (enable/disable) can be specified either with the `CMDFILE STARTUP` Command Line command or using the Startup index of the Command Files dialog.

By default the `STARTUP.CMD` file located in the current project directory is enabled as the current Startup command file.

6.1.2 Reset Command File

The Reset command file is executed by the *Metrowerks debugger* straight after the reset button, menu entry or Command Line command has been selected.

The Reset command file full name and status (enable/disable) can be specified either with the `CMDFILE RESET` Command Line command or using the Reset index of the the Command Files dialog.

By default the `RESET.CMD` file located in the current project directory is enabled as the current Reset command file.

6.1.3 Preload Command File

The Preload command file is executed by the *Metrowerks debugger* right before an application is loaded to the target system through the Target Interface.

The Preload command file full name and status (enable/disable) can be specified either with the `CMDFILE PRELOAD` Command Line command or using the Preload index of the Command Files dialog

By default the PRELOAD.CMD file located in the current project directory is enabled as the current Preload command file.

6.1.4 Postload Command File

The Postload command file is executed by the *Metrowerks debugger* right after an application has been loaded to the target system through the Target Interface.

The Postload command file full name and status (enable/disable) can be specified either with the CMDFILE POSTLOAD Command Line command or using the Postload index of the Command Files dialog.

By default the POSTLOAD.CMD file located in the current project directory is enabled as the current Postload command file.

6.1.5 Vppon Command File

The Vppon command file is executed by the *Metrowerks debugger* right before a "Non Volatile Memory" is erased or right before a file is programmed in "Non Volatile Memory" to the target system through the Target Interface Non Volatile Memory Control dialog (Flash... menu entry) or FLASH PROGRAM/ERASE commands from Flash Programming utilities.

The Vppon command file full name and status (enable/disable) can be specified either with the CMDFILE VPPON Command Line command or using the Vppon index of the Command Files dialog.

By default the VPPON.CMD file located in the current project directory is enabled as the current Vppon command file.

This command file can be used for example to enable a programming voltage by software.

WARNING! This command file is not available for all Target Interfaces.

6.1.6 Vppoff Command File

The Vppoff command file is executed by the *Metrowerks debugger* right after a "Non Volatile Memory" has been erased or right after a file has been programmed in "Non Volatile Memory" to the target system through the Target Interface Non Volatile Memory Control dialog (Flash... menu entry) or FLASH PROGRAM/ERASE commands from Flash Programming utilities.

The Vppoff command file full name and status (enable/disable) can be specified either with the `CMDFILE VPPOFF` Command Line command or using the Vppoff index of the Command Files dialog.

By default the `VPPOFF.CMD` file located in the current project directory is enabled as the current Vppoff command file.

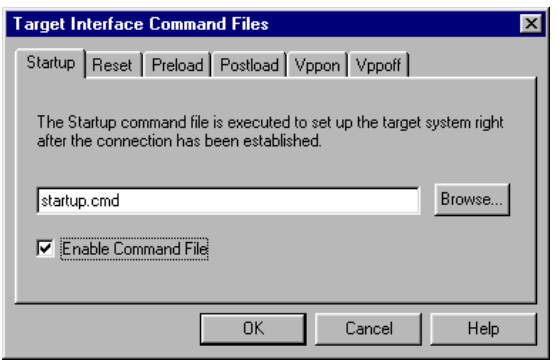
This command file can be used for example to disable a programming voltage by software.

WARNING! This command file is not available for all Target Interfaces.

6.2 Command Files dialog

The Target Interface Command Files dialog can be opened selecting menu entry **"TargetName" > Command Files**. (In this section, **TargetName** is the name of the target, like *SDI*, *Hitex*, *BDIK*, *ICD-12*, *Noral-BDM*, etc.)

Figure 6.1 Target Interface Command Files dialog



Each index of this dialog corresponds to an event on which Target Interface Commands Files can be automatically run from the *Metrowerks debugger*: Startup Command File, Reset Command File, Preload Command File, Vppon Command File, (not available for all targets), Vppoff Command File (not available for all targets), or any other Target Interface specific command file.

The command file in the edit box is executed when the corresponding event occurred.

Using the *Browse* button, you can set up the path and name of the command file.

The *Enable Command File* check box allows to enable/disable a command file on a event. By default, all command files are enabled:

- the default Startup command file is STARTUP.CMD,
- the default Reset command file is RESET.CMD,
- the default Preload command file is PRELOAD.CMD,
- the default Postload command file is POSTLOAD.CMD.
- the default Vppon command file is VPPON.CMD.
- the default Vppoff command file is VPPOFF.CMD.

NOTE	The settings performed in this dialog are stored for a later debugging session in the ["targetName"] section of the PROJECT file using variables CMDFILE0, CMDFILE1, ... CMDFILEn.

6.3 Associated Commands

This section describes the Command Files command which can be used when the Target Interface is set.

The Target Interface specific commands are:

CMDFILE

Those commands can be entered in the command files or in the Command Line component of the *Metrowerks debugger*.

This section describes each command available for the Target Interface. The commands are listed in alphabetical order.

Topic	Description
Short Description	Provides a short description of the command.
Syntax	Specifies the syntax of the command in a EBNF format.
Description	Provides a detailed description of the command and how to use it.
Example	Small example of how to use the command.

CMDFILE

Short Description

Defines a command file path, name and status (enable/disable).

Syntax

```
CMDFILE <file kind> ON|OFF ["<file name and path>"]
and
file kind = STARTUP|RESET|PRELOAD|POSTLOAD|VPPON|VPPOFF
```

Description

The CMDFILE command is to be used set up a command file full name and status (disabled/enabled).

This command allows to perform the same settings than using the Command Files dialog through the Command Line component.

The settings of a command file are stored in the ["targetName"] section of the PROJECT file using variable CMDFILEn.

Example

The list of available command files (and their status) can be get typing CMDFILE without any parameters in the Command Line component.

```
in>CMDFILE
Hitex Target Interface Command Files:
STARTUP ON startup.cmd
RESET ON reset.cmd
PRELOAD ON preload.cmd
POSTLOAD ON postload.cmd
```

The status of the Startup command file can be changed:

```
in>CMDFILE STARTUP OFF "my own startup.cmd"
in>CMDFILE
Hitex Target Interface Command Files:
STARTUP OFF my own startup.cmd
RESET ON reset.cmd
PRELOAD ON preload.cmd
POSTLOAD ON postload.cmd
```

6.4 Associated Environment Variables

This section describes the Command Files dialog environment variables which are used by the Target Interface.

CMDFILEn

These variables are stored in the `["targetName"]` section from the project file.

Listing 6.2 Example of the `[NORAL FLEX BDM]` target section from the project file:

```
[NORAL FLEX BDM]
CMDFILE0=CMDFILE STARTUP ON "startup.cmd"
CMDFILE1=CMDFILE RESET ON "reset.cmd"
CMDFILE2=CMDFILE PRELOAD ON "preload.cmd"
CMDFILE3=CMDFILE POSTLOAD ON "postload.cmd"
CMDFILE4=CMDFILE VPPON ON "vppon.cmd"
CMDFILE5=CMDFILE VPPOFF ON "vppoff.cmd"
```

The following section describes each variable available for the Target Interface. The variables are listed in alphabetical order.

Topic	Description
Short Description	Provides a short description of the variable.
Syntax	Specifies the syntax of the variable in a EBNF format.
Default	Shows the default setting for the variable.
Description	Provides a detailed description of the variable and how to use it.
Example	Small example of how to use the variable.

CMDFILEn

Short Description

Contains a CMDFILE Command Line command to be used to define a command file on a event.

For More Information: www.freescale.com

Syntax

```
CMDFILEn=<command file specified using CMDFILE Command Line command>
```

Default

All command files are enabled by default.

The default Startup command file is STARTUP.CMD,

The default Reset command file is RESET.CMD,

The default Preload command file is PRELOAD.CMD,

The default Postload command file is POSTLOAD.CMD.

The default Vppon command file is VPPON.CMD.

The default Vppoff command file is VPPOFF.CMD.

Description

The CMDFILEn variable specifies a command file definition using CMDFILE Command Line command.If there are four Target Interface Commands Files for the Target Interface, four of those entries should be present.

Those variables are used to store the command files status (enable/disable) and full name specified either with the CMDFILE Command Line command or using the Command Files dialog.

Example

```
CMDFILE0=CMDFILE STARTUP ON "startup.cmd"
CMDFILE1=CMDFILE RESET ON "reset.cmd"
CMDFILE2=CMDFILE PRELOAD ON "preload.cmd"
CMDFILE3=CMDFILE POSTLOAD ON "postload.cmd"
CMDFILE4=CMDFILE VPPON OFF "vppon.cmd"
CMDFILE5=CMDFILE VPPOFF OFF "vppoff.cmd"
```

7

On-Chip Hardware Breakpoint Module

On some HC12 and HCS12 derivatives, an on-chip hardware breakpoint module can be used to set triggers. The *Metrowerks debugger* takes advantage of this embedded hardware breakpoint module to set hardware breakpoints and watchpoint.

To invoke this module, it is necessary to set up the debugger to use this on-chip hardware breakpoint module.

During the first connection, the hardware breakpoints module settings are resolved according to the specified derivative.

Afterwards, if the user change the derivative, it is the user responsibility to setup correctly the hardware breakpoints mechanism for the project.

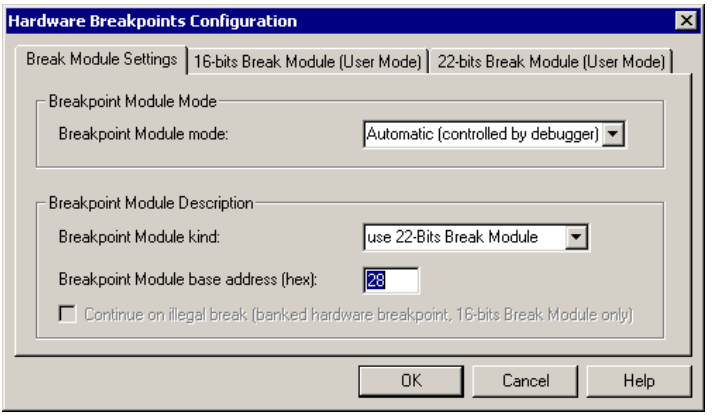
This can be done using the Hardware Breakpoint Configuration dialog.

7.1 Hardware Breakpoint Configuration dialog

Choose **ICD-12 > Set Hardware BP...** menu command. The *Hardware Breakpoint Configuration* dialog, *Break Module Settings* index tab is displayed, as shown in Figure 7.1.

For More Information: www.freescale.com

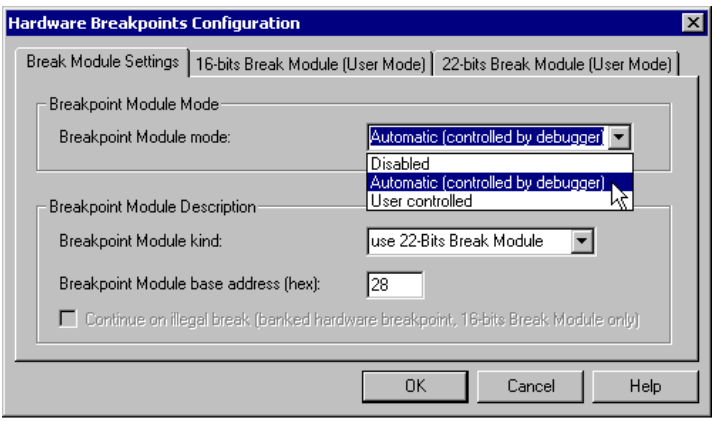
Figure 7.1 Hardware Breakpoint Configuration dialog



Breakpoint Module Mode: The *Mode* combo box allows to select between three different modes: *Disabled*, *Automatic (controlled by debugger)* and *User controlled* (See Figure 7.2).

This dialog allows to set up the hardware breakpoint module of the used *Motorola HC12* or *HCS12* derivative.

Figure 7.2 Hardware Breakpoint Configuration Breakpoint Module mode



NOTE	This feature is available only if the *Motorola HC12* or *HCS12* derivative connected to the debugger through the *P&E Cable12* or *BDM-MULTILINK* has an embedded hardware breakpoint module. Check your MCU documentation

7.1.1 Disabled mode

When the hardware breakpoint module is disabled, it is not possible to set breakpoint in Flash or in EEPROM. It is also not possible to set any watchpoint, even if the application is loaded in RAM.

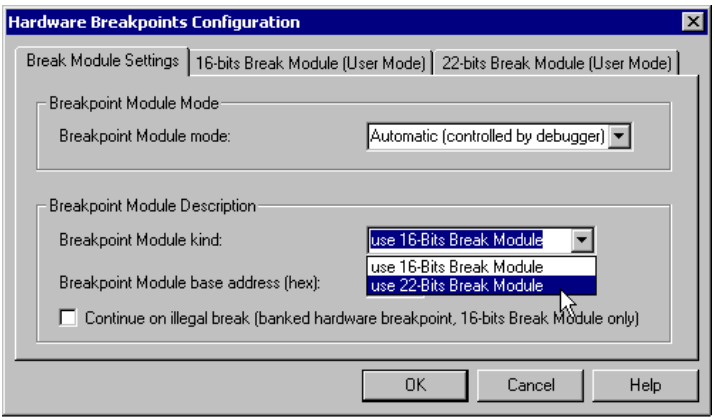
NOTE	Some actions like “stepping over” or “stepping out” use one internal breakpoint and therefore can not be used when debugging in non volatile memory if the hardware breakpoint module is disabled.

7.1.2 Automatic (controlled by debugger) mode

This is the default mode for the debugger.

If the *Automatic (controlled by debugger)* mode is selected, you have the possibility to set up to two breakpoints in *Non Volatile Memory* (or one watchpoint), as shown in Figure 7.3.

Figure 7.3 Module base address edit box



Breakpoint Module kind: select here the hardware breakpoint module supported by the derivative currently connected: “use 16-Bits Break Module” for a *Motorola HC12* derivative and “use 22-Bits Break Module” for a *Motorola HCS12* derivative.

Breakpoint Module base address (hex): in order to set the debugger correctly, the address of the hardware breakpoint module must be set in the *Module base address* edit box. The Module base address is typically **`0x20`** for the *Motorola HC12* derivatives *M68HC912B32*, *M68HC912D60* and *M68HC912DG128*. The Module base address is typically **`0x28`** for the *Motorola HCS12* derivatives.

Continue on illegal break: this feature allows, when using the 16 Bits-break module to debug in banked memory model. The 16 bits break module does not allow to set a breakpoint in bank. To solve this problem, when the debugger stops on a hardware breakpoint, the address if compared to an internal breakpoint list. If the low 16 bits part of the address compare to the low 16 bits part of the address of a set breakpoint, the breakpoint is located in an alternate bank. The debugger then automatically restarts the target.

When those settings are done, any breakpoint which is set in *Non Volatile Memory* is considered by the debugger as an *Hardware Breakpoint*.

If your application is loaded in RAM, breakpoints are software breakpoints. In this case the *Hardware Breakpoint* module gives you the possibility to debug with breakpoints and watchpoint (only one watchpoint is available).

NOTE In Automatic mode, the *Motorola HC12* or *Motorola HCS12* hardware breakpoint modules allow only two breakpoints (or one watchpoint) at the same time. If you are debugging your code in FLASH, you can not set more than two breakpoints or one watchpoint.

Some actions like “stepping over” or “stepping out” use one internal breakpoint and therefore reduce your amount of hardware breakpoint to one.

The M68HC812A4 does not have any Hardware Breakpoint module.

7.1.3 User Controlled mode

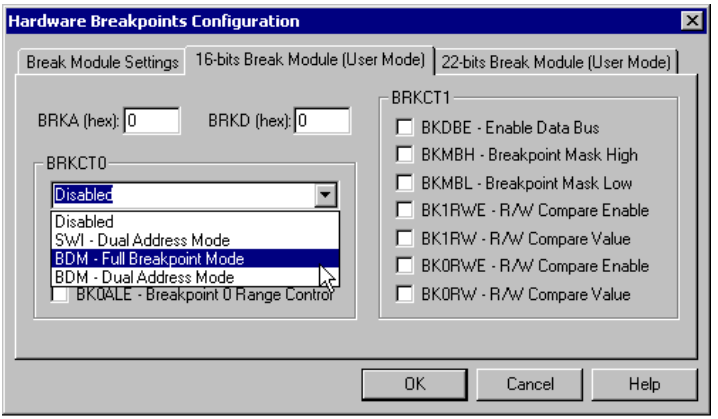
This mode allows you to fully set up the breakpoint module according to Motorola’s documentation.

According to the selected breakpoint module kind selected through the *Breakpoint Module Description* combo box in the Break Module Setup index tab, the 1*6-bits Break Module (User Mode)* or *22-bits Break Module (User Mode)* must be selected (the control are grayed in the *User Mode* index tab if the correct Mode (*User Controlled* and correct breakpoint module kind is not selected).

7.1.3.1 16-bits Break Module (User Mode)

The *16-bits Break Module (User Mode)* index tab allows to set up the hardware breakpoint module of the connected *Motorola HC12* derivative when the *Breakpoint Module mode* is set to *“User controlled”* and the *Breakpoint Module Kind* is set to use *“16-Bits Break Module”*.

Figure 7.4 16-bits Break Module (User Mode) index tab



The following registers can be modified:

- **BRKCT0**: Breakpoint Control Register 0
- **BRKCT1**: Breakpoint Control Register 1
- **BRKA**: Breakpoint Address Register
- **BRKD**: Breakpoint Data Register

For more information about those registers, please refers to your MCU reference manual section *Breakpoints* of the *Background Debug Mode* (*Development Support* part of the manual).

CAUTION When a hardware breakpoint or watchpoint is set in *User controlled* mode, the message displayed in the status bar when the breakpoint or watchpoint is reached is ILLEGAL_BP.

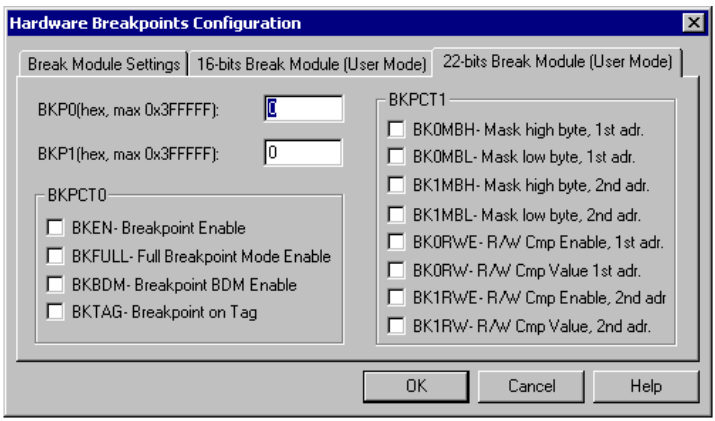
If the control point set is a breakpoint, it is needed to perform a single step before running again the target otherwise the target will endlessly break on the same address bus access.

7.1.3.2 22-bits Break Module (User Mode)

The *22-bits Break Module (User Mode)* index tab allows to set up the hardware breakpoint module of the connected *Motorola HCS12* derivative when the *Breakpoint*

Module mode is set to *"User controlled"* and the *Breakpoint Module Kind* is set to use *"22-Bits Break Module"*.

Figure 7.5 22-bits Break Module (User Mode) index tab



The following registers can be modified:

- **BKPCT0**: Breakpoint Control Register 0
- **BKPCT1**: Breakpoint Control Register 1
- **BKP0**: Breakpoint Address Register
- **BKP1**: Breakpoint Data Register

For more information about those registers, please refers to your MCU reference manual section *Breakpoints* of the *Background Debug Mode* (*Development Support* part of the manual).

CAUTION When a hardware breakpoint or watchpoint is set in *User controlled* mode, the message displayed in the status bar when the breakpoint or watchpoint is reached is ILLEGAL_BP.

If the control point set is a breakpoint, it is needed to perform a single step before running again the target otherwise the target will endlessly break on the same address bus access.

7.2 Associated Commands

The following sections describe the Hardware Breakpoint Settings Command Line commands which are used by the Target Interface. These variables are:

HWBPM

Those commands can be entered in the Target Interface associated command files or in the Command Line component of the debugger.

The Hardware Breakpoint Settings commands which are used by the Target Interface are described as shown in the following table.

Topic	Description
Short Description	Provides a short description of the command.
Syntax	Specifies the syntax of the command in a EBNF format.
Description	Provides a detailed description of the command and how to use it.
Example	Small example of how to use the command.

The following sections describe each command related to the Banked Memory Location available for the Target Interface. The variables are listed in alphabetical order.

HWBPM

Short Description

Hardware Breakpoints module usage settings.

Syntax

```
HWBPM
HWBPM MODE <MODE> BPM16BITS|BPM22BITS <module adr.> [SKIP_OFF|SKIP_ON]
with MODE = DISABLED|AUTOMATIC|USER

HWBPM SET16BITS <BRKCT0 value> <BRKCT1 value> <BRKA value> <BRKD value>
HWBPM SET22BITS <BKPCT0 value> <BKPCT1 value> <BKP0 value> <BKP1 value>
```

```
HWBPM REMAP_22BITS RANGE <start address> <end address> <mask>
HWBPM REMAP_22BITS DISPLAY
HWBPM REMAP_22BITS MCUID_DEFAULT
HWBPM REMAP_22BITS DELETE <range number>
```

Description

The command `HWBPM` allows to set up the debugger to work with the on chip hardware breakpoints dialog.

Use HWBPM with no parameters to get the current breakpoints settings.

Using HWBPM MODE ..., you can set up which module to use, the usage the debugger will do of the on-chip hardware breakpoint module, the on-chip module address, etc.. This command will have the same effect than using the *Break Modules Settings* index tab in the Hardware Breakpoint Configuration dialog.

Using HWBPM SET16BITS ... command will have the same effect than using the *16-bits Break Module (User Mode)* index tab in the Hardware Breakpoint Configuration dialog. Parameters set up through this command are only relevant when the *User controlled* mode is active and the 16 bits break module is used.

Using HWBPM SET22BITS ... command will have the same effect than using the *22-bits Break Module (User Mode)* index tab in the Hardware Breakpoint Configuration dialog. Parameters set up through this command are only relevant when the *User controlled* mode is active and the 22 bits break module is used.

NOTE	The hardware breakpoints settings are stored in the ["targetName"] section of the PROJECT file using variable HWBPMn.

The HWBPM REMAP_22BITS commands are used, for the 22-bits module, to perform remapping of pages, in order to be able to set breakpoints in non banked memory areas when using this on-chip break module. When selecting a derivative, this command is used by the debugger to set up the corresponding remapping needed for the specified derivative.

- HWBPM REMAP_22BITS DISPLAY display all the currently set remapping, for the currently set derivative.
- HWBPM REMAP_22BITS RANGE allows to specify that the prefix <mask> must be used to set a hardware breakpoint in range <start address> <end address>
- HWBPM REMAP_22BITS MCUID_DEFAULT allows to retrieve the derivative default setting (in case it has been modified using HWBPM REMAP_22BITS RANGE ... or HWBPM REMAP_22BITS DELETE)

- HWBPM REMAP_22BITS DELETE <range number> allows to delete a specific range. The range number is displayed when using HWBPM REMAP22BITS DISPLAY.

NOTE The range remapping are stored in the ["targetName"] section of the PROJECT file using variable HWBPD_MCUIDnnn_BKPT_REMAPn.

Example

The Hardware Breakpoints mechanism settings can be get typing HWBPM without any parameters in the Command Line component.

```
in>HWBPM
Hardware Breakpoints Module Settings:
Module kind:    22BITS
Module mode:    Automatic
Module address: 0x28
Skip illegal BP (16bits only): off
HWBPM 16 bits: BRKCT0: 0x0 BRKCT1: 0x0 BRKA: 0x0 BRKD: 0x0
HWBPM 22 bits: BKPCT0: 0x0 BKPCT1: 0x0 BKP0: 0x0 BKP1: 0x0
```

The current Module mode can be modified to User controlled and the used on-chip hardware breakpoint module to the 16-bits one (relevant only if present on the hardware):

```
in>HWBPM MODE USER BPM16BITS 0x20 SKIP_OFF
in>HWBPM
Hardware Breakpoints Module Settings:
Module kind:    16BITS
Module mode:    User Defined
Module address: 0x20
Skip illegal BP (16bits only): off
HWBPM 16 bits: BRKCT0: 0x0 BRKCT1: 0x0 BRKA: 0x0 BRKD: 0x0
HWBPM 22 bits: BKPCT0: 0x0 BKPCT1: 0x0 BKP0: 0x0 BKP1: 0x0
```

Enter values in the on-chip breakpoint module registers:

```
in>HWBPM SET16BITS 0xa4 0x0 0xc004 0x0
in>HWBPM
Hardware Breakpoints Module Settings:
Module kind:    16BITS
Module mode:    User Defined
Module address: 0x20
Skip illegal BP (16bits only): off
HWBPM 16 bits: BRKCT0: 0xa4 BRKCT1: 0x0 BRKA: 0xc004 BRKD:
0x0
```

```
HWBPM 22 bits: BKPCT0: 0x0 BKPCT1: 0x0 BKP0: 0x0 BKP1: 0x0
```

Display the currently set remapping:

```
in>HWBPM REMAP_22BITS DISPLAY
HWBPM Remappings for 0x3CA:
Range0:    0x4000..0x7FFF mask: 0x3e
Range1:    0xC000..0xFFFF mask: 0x3f
```

Add a new remapping:

```
in>HWBPM REMAP_22BITS RANGE 0x8000 0xbfff 0x47
in>HWBPM REMAP_22BITS DISPLAY
HWBPM Remappings for 0x3CA:
Range0:    0x4000..0x7FFF mask: 0x3e
Range1:    0xC000..0xFFFF mask: 0x3f
Range2:    0x8000..0xBFFF mask: 0x47
```

Delete a remapping:

```
in>HWBPM REMAP_22BITS DELETE 1
in>HWBPM REMAP_22BITS DISPLAY
HWBPM Remappings for 0x3CA:
Range0:    0x4000..0x7FFF mask: 0x3e
Range1:    0x8000..0xBFFF mask: 0x47
```

Retrieve the default remapping for the currently set derivative:

```
in>HWBPM REMAP_22BITS MCUID_DEFAULT
in>HWBPM REMAP_22BITS DISPLAY
HWBPM Remappings for 0x3CA:
Range0:    0x4000..0x7FFF mask: 0x3e
Range1:    0xC000..0xFFFF mask: 0x3f
```

7.3 Associated Environment Variables

The following sections describe the Hardware Breakpoint Settings environment variables which are used by the Target Interface. These variables are:

HWBPD_MCUIDnnn_BKPT_REMAPn

HWBPMn

These variables are stored in the ["targetName"] section from the project file.

Example of the [ICD12] target section from a project file:

```
[ICD12]
HWBPM0=HWBPM MODE AUTOMATIC BPM16BITS 0x28 SKIP_OFF
HWBPM1=HWBPM SET16BITS 0x0 0x0 0x0 0x0
HWBPM2=HWBPM SET22BITS 0x0 0x0 0x0 0x0
```

```
HWBPD_MCUID3C6_BKPT_REMAP0=HWBPM REMAP_22BITS RANGE 0x4000 0x7FFF 0x3E
HWBPD_MCUID3C6_BKPT_REMAP1=HWBPM REMAP_22BITS RANGE 0xC000 0xFFFF 0x3F
HWBPD_MCUID3C7_BKPT_REMAP0=HWBPM REMAP_22BITS RANGE 0x4000 0x7FFF 0x3E
HWBPD_MCUID3C7_BKPT_REMAP1=HWBPM REMAP_22BITS RANGE 0xC000 0xFFFF 0x3F
HWBPD_MCUID3CA_BKPT_REMAP0=HWBPM REMAP_22BITS RANGE 0x4000 0x7FFF 0x3E
HWBPD_MCUID3CA_BKPT_REMAP1=HWBPM REMAP_22BITS RANGE 0xC000 0xFFFF 0x3F
```

The Hardware Breakpoint Settings variables which are used by the Target Interface are described as shown in the following table.

Topic	Description
Short Description	Provides a short description of the variable.
Syntax	Specifies the syntax of the variable in a EBNF format.
Default	Shows the default setting for the variable.
Description	Provides a detailed description of the variable and how to use it.
Example	Small example of how to use the variable.

The following sections describe each variable available for the Target Interface. The variables are listed in alphabetical order.

HWBPD_MCUIDnnn_BKPT_REMAPn

Short Description

Contains a HWBPM Command Line command to be used to set up the Hardware Breakpoint Remapping.

Syntax

```
HWBPD_MCUIDnnn_BKPT_REMAPn=<one HWBPM REMAP22BITS Command Line command>
```

Default

Defaults settings are retrieved according to the derivative from a common ini file.

Description

The HWBPD_MCUIDnnn_BKPT_REMAPn variable specifies a command file definition using HWBPM REMAP22BITS Command Line command.

The variable name depends on the derivative MCU-ID and on the remapping range number.

Those variables are used to store the current Hardware Breakpoints Module remapping settings specified with the HWBPM REMAP22BITS Command Line command.

Example

```
HWBPD_MCUID3C6_BKPT_REMAP0=
          HWBPM REMAP_22BITS RANGE 0x4000 0x7FFF 0x3E
```

HWBPMn

Short Description

Contains a HWBPM Command Line command to be used to set up the Hardware Breakpoint Settings support.

Syntax

```
HWBPMn=<one HWBPM Command Line command>
```

Default

Defaults settings are retrieved according to the derivative from a common ini file.

Description

The HWBPMn variable specifies the configuration of the Hardware Breakpoints module using HWBPM Command Line command. Three entries should be present in the project file.

Those variables are used to store the current Hardware Breakpoints Module settings specified either with the HWBPM Command Line command or through the Hardware Breakpoint Configuration dialog.

Example

```
HWBPM0=HWBPM MODE AUTOMATIC BPM16BITS 0x28 SKIP_OFF
HWBPM1=HWBPM SET16BITS 0x0 0x0 0x0 0x0
HWBPM2=HWBPM SET22BITS 0x0 0x0 0x0 0x0
```

8

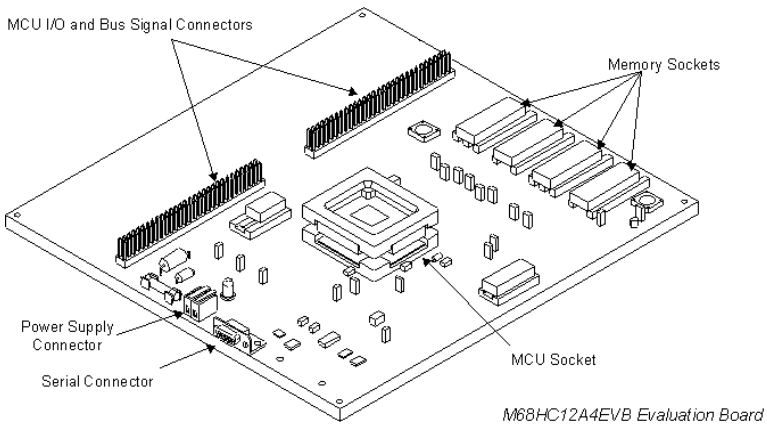
Running Standard EVBs with a BDM pod

8.1 Connection between a BDM pod and an EVB board

Motorola's Evaluation Boards also called "*EVB*", and other standard boards from other manufacturers are equipped with a BDM connector and therefore can be connected to any BDM interface.

Figure 8.1 represents a typical layout of an evaluation board.

Figure 8.1 Layout of a Motorola Evaluation Board.



Some boards have a 6-pin BDM male connector and some have a 10-pin BDM male connector.

For More Information: www.freescale.com

On the ribbon cable, the red wire is the wire number 1. Pin 1 of the male **BDM-in** connector of EVB boards is identified by the number "1" written on the board or by a square pad (copper side of the board) under the connector.

8.2 MC68HC812A4EVB Evaluation Board

This Evaluation Board for the HC12 from Motorola is ready to work in BDM mode. It supports a HC12A4 processor.

This procedure is extracted from the *HC12A4EVBUM/D* hardware manual from Motorola (*Appendix F*), to run the MC68HC812A4EVB with a BDM interface:

1.Remove the jumper on header **W11** from **CSD**.

2.Move the **CSP0** jumper on **W11** to pins 2-3.

3.Remove the **BKGD** jumper (**W30**).

4.The **MODA** jumper (**W34**) must short-cut the pin 1 and pin 2.

5.The **MODB** jumper (**W42**) must short-cut the pin 1 and pin 2.

6.Software setup to be inserted in the Startup Command File and in the Reset Command File of your current project:

```
wb 0x000B 0xF0 //MODE: forcing the Normal Exp Wide mode
wb 0x000B 0xF0 //    : this forcing must be done twice!!
wb 0x0011 0x00 //INITRG
wb 0x0010 0x08 //INITRM
wb 0x0012 0x11 //INITEE, move on-chip EEPROM from $F000
               //        to $1000
```

NOTE	You will notice that the hardware setup steps 4 and 5 (given by Motorola) force the HC12A4 operating mode to Single Chip, this to enable to start the background debugger. Then you must force by software (as shown above in the `RESET.CMD` file) the Expanded Wide mode to be able to access the on-board 16 Kilobytes RAM mapped from `$C000` to `$FFFF`.

After the setup given above, the new HC12A4 memory map is:

$0000-$01FF, CPU registers, on-chip (MCU)

$0800-$0BFF, user data area, 1 Kilobyte on-chip RAM (MCU)

$1000-$1FFF, user code area, 4 Kilobyte on-chip EEPROM (MCU)

$C000-$FFFF, user code/data area, 16 Kilobytes external RAM (U4, U5A)

8.3 MC68HC912B32EVB Evaluation Board

This Evaluation Board for the HC12 from Motorola is read to work in BDM mode. It supports a *M68HC912B32* processor.

To run a BDM pod with the *MC68HC912B32EVB*, the jumper settings are the default of the board.

8.4 MC68HC912D60EVB Evaluation Board

This Evaluation Board for the HC12 from Motorola is read to work in BDM mode. It supports a *M68HC912D60* processor.

To run a BDM pod with the *MC68HC912D60EVB*, the jumper settings are the default of the board.

The BDM connector of the BDM cable must be connected to the BDM connector of the board (**J17**).

8.5 MC68HC912DG128EVB Evaluation Board

This Evaluation Board for the HC12 from Motorola is ready to work in BDM mode. It supports a *M68HC912DG128* processor.

To run a BDM pod with the *MC68HC912DG128EVB*, the jumper settings are the default of the board.

The BDM connector of the BDM cable must be connected to the BDM connector of the board (**J16**).

8.6 EVB912DP256 Evaluation Board

This Evaluation Board for the HCS12 from Motorola is ready to work in BDM mode. It supports a *MC9S12DP256* processor.

To run a BDM pod with the *EVB912DP256*, the jumper settings are the default of the board.

The BDM connector of the BDM cable must be connected to the BDM IN connector of the board (**J19**).

8.7 Axiom CML12S-DP256

CodeWarrior does not support the "Axiom MON12" monitor. But the board can be monitored by using the BDM interface.

To run a BDM pod with the *CML12S-DP256*, make sure the MODC jumper is installed. Without it the BDM application may cause communication problems with the host.

All other jumpers can be left on default settings.

WARNING!	By Flashing the board with an application the "Axiom MON12" monitor will be overwritten. It is possible to backup the monitor by flashing the monitor to the chip again. The monitor program can be found on the Axiom CD.

Index

Symbols

.abs 28

Numerics

16-bits Break Module (User Mode) 81
22-bits Break Module (User Mode) 82

A

Associated Environment Variables 67
At Connection, detect if the target is running 31
Axiom CML12S-DP256 94

B

Banked Memory Location dialog 51
Banked Memory Location Target commands 55
BDM 91
BDM connector 24, 25
BDM MULTILINK 32
BDM Multilink 62
BDMAREASUPPORT 45
BDMAutoSpeed 38
BDMClockSpeed 39
BREAKPOINT 35
Breakpoint 21, 79, 81
    BREAKPOINT 35
    Message 35

C

CABLE12 32
Cable12 62
CABLE12HS 32
CLKSW 30
clock divider 32
CMDFILE 74
CMDFILEn 75
COMDEV 40
Command Files 28, 69
Commands 55, 66, 73, 84
Communication 28
Communication Device 30
Communication device 30
Communication Device Specification 30
Connect 28
Connection 24
Coverage 23

D

DETECTRUNNING 41
Display Bank Memory Location dialog at connection 55
DPAGE 52
DPAGE Banked Memory Area 52

E

Environment variables 36
EPAGE 53
EPAGE Banked Memory Area 53
EVB 91
EVB912DP256 94

F

Flash Programming 33
Flash programming 71
Flash... 29

H

HALTED 35
Hardware Breakpoint 77
Hardware Breakpoint module
    Automatic (controlled by debugger mode) mode 79
    Disabled mode 79
    User controlled mode 81
Hardware Breakpoints SettingTarget commands 84
Hardware Connection 24
HCS12 Unsecure Target commands 66
Highlights 8
How To
    Loading an Application in Flash EEPROM 13
    Setting Breakpoints in Non Volatile Memory 20
HWBPM 85
HWBPM MODE 85
HWBPM REMAP_22BITS 85
HWBPM REMAP_22BITS DELETE 86
HWBPM REMAP_22BITS DISPLAY 85

For More Information: www.freescale.com

HWBPM REMAP_22BITS MCUID_DEFAULT 85
HWBPM REMAP_22BITS RANGE 85
HWBPM SET16BITS 85
HWBPM SET22BITS 85
HWBREAKONLY 45

I

I/O 23
ICD-12
    Default target 36
    Menus 36
ICD-12 > Command Files 28
ICD-12 > Communication... 28
ICD-12 > Connect... 28
ICD-12 > Flash... 29
ICD-12 > Load... 28
ICD-12 > Reset 28
ICD-12 > Set Bank... 29
ICD-12 > Set BDM Speed... 28
ICD-12 > Set Hardware BP... 29
ICD-12 > Set MCU Type... 28
ICD-12 > Unsecure... 29
icd12.tgt 36
INITRG 48
IO_DELAY_COUNT 32, 62

L

LISTALLMCUIDS 46
Load... 28
Loading an application 28
LPT1 10
LPT2 10

M

MC68HC812A4EVB 92
MC68HC912B32EVB 93
MC68HC912D60EVB 93
MC68HC912DG128EVB 93
MCU Selection dialog 27
MCUID 42
MDSEMCU.INI 31, 32
Metrowerks Debugger
    Status Bar 34
Module base address 80

For More Information: www.freescale.com

N

Non Volatile Memory 71

O

Overview 7

P

P&E Microcomputer Systems 23
PLL 30
Postload command file 71
Power supply 25
PPAGE 52
PPAGE Banked Memory Area 52
Preload command file 70
Profiling 23
PROJECT File 36
PROJECT.INI 25
PROTOCOL 47
Protocol 25

R

READY 34
REGBLKADR 47
Requirements 8
RESET 48
Reset 28
Reset command file 70
RUNNING 34
running 31

S

Set Bank... 29
Set BDM Speed... 28
Set CLKSW bit in BDM control register 30
Set Hardware BP... 29
Set MCU Speed 32
Set MCU Type... 28
SETCLKSW 42
Show Protocol 30
SHOWPROT 43
Simulation 23
Startup command file 70
Status Message 34
    HALTED 35

For More Information: www.freescale.com

READY 34
Reset 35
RUNNING 34
STEPPED 35
STEPPED OVER 35
Stepping Message 35
STEPPED 35
STOPPED 35
TRACED 35
STOPPED 35

T

Target Command Files 69
Target commands 73
Target Interface Associated Command Files 69
Target Interface Command Files dialog 72
Target Interface Dialogs 51
Target Interface Command Files dialog 72
TRACED 35

U

unit_12z.dll 24
Unsecure Command File 63
Unsecure derivative dialog box 61
Unsecure HCS12 derivatives 61
Unsecure... 29

V

Variable 36
Variables 67
Various index tab 54
Vppoff command file 71
Vppon command file 71

W

WATCHPOINT 35
Watchpoint
WATCHPOINT 35

For More Information: www.freescale.com